NATIONAL INSTRUMENTS™
LabVIEW™

VXI VI Reference Manual

Worldwide Technical Support and Product Information

`www.ni.com`

National Instruments Corporate Headquarters

11500 North Mopac Expressway   Austin, Texas 78759-3504   USA   Tel: 512 794 0100

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Calgary) 403 274 9391, Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24,
Germany 089 741 31 30, Greece 30 1 42 96 427, Hong Kong 2645 3186, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico (D.F.) 5 280 7625,
Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466, New Zealand 09 914 0488, Norway 32 27 73 00,
Poland 0 22 528 94 06, Portugal 351 1 726 9011, Singapore 2265886, Spain 91 640 0085,
Sweden 08 587 895 00, Switzerland 056 200 51 51, Taiwan 02 2528 7227, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix. To comment on the
documentation, send e-mail to `techpubs@ni.com`

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 document 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

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

Trademarks

LabVIEW™, National Instruments™, MITE™, ni.com™, and NI-VXI™ are trademarks of National Instruments Corporation.

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

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Contents

Chapter 3
Word Serial Commander Protocol VIs

Chapter 4
Word Serial Servant Protocol VIs

Chapter 5
Low-Level VXIbus Access VIs

Chapter 6
High-Level VXIbus Access VIs

Chapter 7
Local Resource Access VIs

Chapter 8
Shared Memory Access VIs

Chapter 9
VXI Signal VIs

Chapter 10
VXI Interrupt VIs

Chapter 11
VXI Trigger VIs

Chapter 12
System Interrupt Handler VIs

Chapter 13
VXIbus Extender VIs

Appendix A
Error Codes

Appendix B
Technical Support Resources

Glossary

Figures

About This Manual

The *LabVIEW VXI VI Reference Manual* describes the VXI virtual instruments (VIs) for LabVIEW.

This manual supplements your *LabVIEW User Manual*, and you should be familiar with the material in that manual. You also should be familiar with the operation of LabVIEW, your computer, and your computer's operating system.

Conventions

The following conventions appear in this manual:

»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
	This icon denotes a note, which alerts you to important information.
bold	Bold text denotes items that you must select or click on in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.
Platform	Text in this font denotes a specific platform and indicates that the text following it applies only to that platform.

Each VXI VI description in this manual displays an icon before the parameter name to designate its data type. These icons are illustrated and defined in the following table.

Control	Indicator	Data Type
I16	**I16**	Integer
I32	**I32**	Long Integer
U16	**U16**	Unsigned Integer
U32	**U32**	Unsigned Long Integer
abc	**abc**	String
TF	**TF**	Boolean
[U32]	**[U32]**	Array of Unsigned Long Integer
[U16]	**[U16]**	Array of Unsigned Integer
[U8]	**[U8]**	Array of Unsigned Character
[I16]	**[I16]**	Array of Integer
[TF]	**[TF]**	Array of Boolean
		Cluster

Related Documentation

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

- *LabVIEW User Manual*
- *LabVIEW Measurements Manual*
- *Getting Started with LabVIEW*
- *LabVIEW Help,* which you launch by selecting by selecting **Help»Contents and Index**
- *LabVIEW Tutorial* **(Windows only),** which you launch from the LabVIEW dialog box
- *LabVIEW Quick Reference Card*

- *LabVIEW Release Notes*
- *LabVIEW Upgrade Notes*
- The getting started or user manuals for the VXI boards you use
- *IEEE Standard for a Versatile Backplane Bus: VMEbus*, ANSI/IEEE Standard 1014-1987
- VXI-1, *VXIbus System Specification*, Rev. 1.4, VXIbus Consortium
- VXI-6, *VXIbus Mainframe Extender Specification*, Rev. 1.0, VXIbus Consortium
- *NI-VXI Programmer Reference Manual*
- *NI-VXI User Manual*

1

Introduction

This manual is a companion guide to the *NI-VXI Programmer Reference Manual* that came with your VXI hardware. With the exception of the LabVIEW Handler VIs, which correspond to the SetHandler and DefaultHandler functions, every VI in this manual corresponds directly with the function of the same name in the *NI-VXI Programmer Reference Manual*.

Read the *NI-VXI User Manual*, as well as this chapter, to give you an overview of the VXIbus and NI-VXI. Once you are familiar with this material, you can begin to write your program. The beginning of each function chapter in this book contains a general overview of the operations of the class of functions in the chapter, as well as a brief description of each function. You should become familiar with the entire class of functions you are using, so you know which ones are necessary to perform your tasks efficiently.

Refer to the README.TXT file in your NI-VXI directory to obtain the latest information about your software, as well as platform-specific information regarding your hardware.

VXIbus Overview

This section introduces some of the concepts from the VXIbus specification.

VXI Devices

A VXI device has a unique logical address, which you use to find or access the device in the VXI system. This logical address is comparable to a GPIB device address. Because VXI uses an 8-bit logical address, you can have up to 256 VXI devices in a VXI system.

Each VXI device must have a specific set of registers called *configuration registers*. These registers are located in the upper 16 KB of the 64-KB A16 VXI address space, as shown in Figure 1-1. The logical address of a VXI device determines the location of the configuration registers of the device in the 16-KB area reserved by VXI.
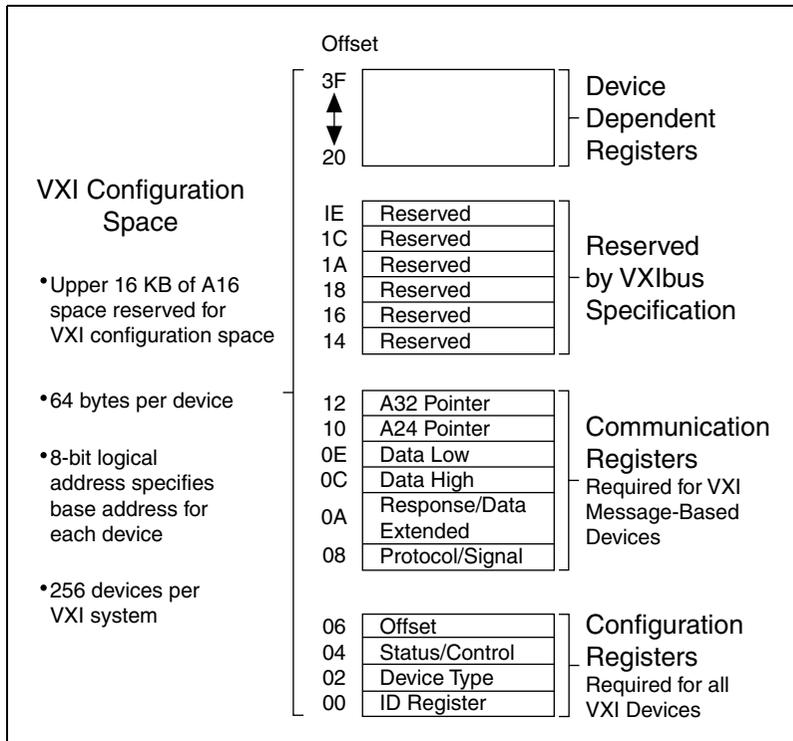
Figure 1-1. VXI Configuration Registers

Register-Based Devices

VXI configuration registers, which are required for all VXI devices, help the system identify each VXI device, its type, model and manufacturer, address space, and memory requirements. A VXIbus device with only this minimum level of capability is called a *Register-Based device*. With this common set of configuration registers, the centralized Resource Manager (RM), which is essentially a software module, can perform automatic system and memory configuration when the system is initialized.

Message-Based Devices

In addition to Register-Based devices, the VXIbus specification also defines *Message-Based devices*, which must have *communication registers* in addition to the configuration registers. All Message-Based VXIbus devices, regardless of the manufacturer, can communicate at a minimum level using the VXI-specified Word Serial Protocol. At this minimum level,

you can establish higher-performance communication channels, such as
shared-memory channels to take advantage of the VXIbus bandwidth
capabilities.

Word Serial Protocol

The VXIbus Word Serial Protocol is a standardized message-passing
protocol. This protocol functions much like the IEEE 488 protocol, which
transfers data messages to and from devices one byte (or word) at a time.
Thus, VXI Message-Based devices communicate in a fashion very similar
to IEEE 488 instruments. In general, Message-Based devices typically
contain some level of local intelligence that uses or requires a high level of
communication.

All VXI Message-Based devices must use Word Serial Protocol and
communicate in a standard way. Use the Word Serial Commander VIs to
communicate with message-based devices.

✏️ **Note**   In this manual, italic marks also denote Word Serial commands and queries.

Commander/Servant Hierarchies

The VXIbus specification defines a Commander/Servant communication
protocol so you can construct hierarchical systems using conceptual layers
of VXI devices. This structure can be compared to an inverted tree. A
Commander is any device in the hierarchy with one or more associated
lower level devices, or Servants. A *Servant* is any device in the subtree of
a Commander. A device can be both a Commander and a Servant in a
multiple-level hierarchy.

A Commander has exclusive control of the communication registers of
its immediate Servants (one or more). Any VXI module has only one
Commander. Commanders communicate with Servants through the
communication registers of the Servants using the Word Serial Protocol.
Servants communicate with their Commander, responding to the Word
Serial commands and queries from their Commander through the Word
Serial Protocol. Servants can also communicate asynchronous status and
events to their Commander through hardware interrupts, or by writing
specific signals directly to the Signal register of their Commander.

Although the Word Serial Protocol is reserved for Commander/Servant
communications, two VXI devices can establish peer-to-peer
communication through a specified shared-memory protocol or by
writing specific messages directly to the Signal register of the device.

Interrupts and Asynchronous Events

Servants can communicate asynchronous status and events to their Commander through hardware interrupts or by writing specific messages (signals) directly to the hardware Signal register of their Commander. Devices that are not bus masters always transmit such information through interrupts, whereas, devices that have bus master capability either can use interrupts or send signals. Some devices can receive only signals, but others might be only interrupt handlers.

The VXIbus specification defines Word Serial commands so that a Commander can understand the capabilities of its Servants and configure them to generate interrupts or signals in a particular way. For example, a Commander can instruct its Servants to use a particular interrupt line, send signals rather than generate interrupts, or configure the reporting of only certain status or error conditions.

VXI Handler VIs Overview

A VXI handler is a user routine that the system executes when some event occurs in the VXI interface. For example, you can set a VXI interrupt handler to execute when an interrupt is asserted.

You can use VXI VIs to create a handler as part of your LabVIEW diagram. The handler can use any LabVIEW VI, including other VXI VIs. The VIs you use to create these handlers are called the Handler VIs.

The Handler VIs are analogous to the SetHandler and DefaultHandler calls in the *NI-VXI Programmer Reference Manual*. The inputs to the Handler VIs are the same as the parameters to the SetHandler functions. The outputs to the Handler VIs are the same as the parameters to the DefaultHandler functions.

The Handler VIs are implemented using LabVIEW occurrences. Because an occurrence is a *one-shot* event (the dependent VIs are executed only once), you must put the Handler VI and your handler in a while loop structure.

Figure 1-2 shows an example of a LabVIEW interrupt handler, which displays the dialog box whenever an interrupt of a certain level is asserted. The VXI VIs store the pending interrupts so that the handler executes once for each handler.

Figure 1-2. Example of an Interrupt Handler

Passing the value 3 to a Set Handler VI tells the VXI VI library to begin queueing occurrences for later handling by a Handler VI.

Note Even on platforms where VXI user handlers normally are executed at interrupt time (such as Macintosh and Windows), the LabVIEW user handler diagram is not executed at interrupt time.

VXI VI Library Overview

The *NI-VXI Programmer Reference Manual* gives explanations of the NI-VXI functions. The material found in that manual applies equally to the NI-VXI LabVIEW VXI VIs, and you can use it as a reference in conjunction with this document. The VIs are divided into the following groups:

- *System Configuration VIs* initialize the NI-VXI interface at the lowest level. In addition, the System Configuration VIs can retrieve or modify device configuration information.

- *Word Serial Commander Protocol VIs* are used by Word Serial Commanders to communicate with a Message-Based Servant device using the Word Serial, Longword Serial, or Extended Longword Serial protocols. Word Serial is the minimal mode of communication between VXI Message-Based devices. These VIs can perform command/query sending and buffer reads/writes.

- *Word Serial Servant Protocol VIs* give Message-Based Servant devices all of the necessary capabilities to communicate with the

Message-Based Commander of the local CPU (the device on which the NI-VXI interface resides) using the Word Serial, Longword Serial, or Extended Longword Serial protocols. These capabilities include command/query handling and buffer read/writes.

- *Low-Level VXIbus Access VIs* offer the fastest access method for directly reading from or writing to any of the VXIbus address spaces. Use them when execution speed is critical.

- *High-Level VXIbus Access VIs* are similar to the Low-Level VXIbus Access VIs, but these VIs give you direct access to the VXIbus address spaces. You can use these VIs to read, write, and move blocks of data between any of the VXIbus address spaces. You can specify any VXIbus privilege mode or byte order. The VIs trap and report Bus Errors. The High-Level VXIbus Access VIs are easy-to-use. You should use them when execution speed is not crucial.

- *Local Resource Access VIs* offer access to miscellaneous local resources such as the VXI register set of local CPU, Slot 0 MODID operations (when the local device is configured for Slot 0 operation), and the VXI shared RAM of the local CPU. These VIs are useful for shared memory type communication, for non-Resource Manager operation (when the local CPU is not the Resource Manager), and for debugging purposes.

- *Shared Memory Access VIs* offer access to local CPU-shared RAM.

- *VXI Signal VIs* are used by VXI bus masters to interrupt another device. The value written to a Signal register of a device has the same format as the status/ID value returned during a VXI interrupt acknowledge cycle. You can route VXI signals to the default interrupt service routine or place them into a global signal queue. VXI signal VIs can specify the signal routing, manipulate the global signal queue, and wait for a particular signal value (or set of values) to be received.

- *VXI Interrupt VIs* let you process individual VXI interrupt status/IDs as VME status/IDs, VXI status/IDs, or VXI signals. By default, status/IDs are processed as VXI signals. VXI interrupt VIs can specify the status/ID processing method and can assert specified VXI interrupt lines with a specified status/ID value.

- *VXI Trigger VIs* are a standard interface to source and accept any of the VXIbus TTL or ECL trigger lines. These VIs can also detect acknowledgements from the accepting device and send the acknowledgements back to the sourcing device. You can use these VIs as configuration tools for signal conditioning and routing trigger lines, and for configuring the settings of the trigger inputs and outputs as well as the National Instruments Trigger Interface Chip (TIC) counter and

tick timers. VXI trigger VIs support all VXI-defined trigger protocols; the actual capabilities depend on the specific hardware platform.

- *System Interrupt Handler VIs* let you handle the system interrupt conditions. These conditions include Sysfail, ACfail, Sysreset, BusError, and SoftReset interrupts.

- *VXIbus Extender VIs* allow you to dynamically reconfigure multiple-mainframe transparent mapping of the VXI interrupt lines, TTL triggers, ECL triggers, and utility bus signals. The National Instruments Resource Manager configures the mainframe extenders with settings based on user-modifiable configuration files.

Refer to `labview\examples\instr\smplvxi.llb` for example VXI VIs.

Multiple Mainframe Support

The NI-VXI functions described in this manual support multiple mainframes both in external CPU configurations and embedded CPU configurations. The Startup Resource Manager supports one or more mainframe extenders and configures a single- or multiple-mainframe VXI/VME system. Refer to the *VXIbus Mainframe Extender Specification*, Revision 1.3 or later, for more information about multiple mainframe systems.

Extender and Controller Parameters

In NI-VXI, some functions require a parameter named **extender** or **controller**. Because some extenders act as controllers, there is often confusion concerning what logical addresses should be passed to these functions.

The **extender** parameter is the logical address of a mainframe extender on which the function should be performed. Usually, functions with an **extender** parameter involve the mapping of interrupt lines or trigger lines into or out of a frame. The **controller** parameter is the logical address of an embedded, external, extending, or remote controller. Usually, functions with a **controller** parameter involve sourcing or sensing particular interrupts or triggers in a frame. According to the definitions of the different types of controllers, the only valid logical addresses for the **controller** parameter are:

- The external or embedded controller on which the program is running.

- A parent-side VXI-MXI-2 or VME-MXI-2 in a frame.

Most functions that take a **controller** parameter allow you to pass (–1) as the logical address. This selects the default controller for the system. Notice that the following factors determine the default controller:

- If the program runs on an embedded controller, the default controller is the embedded controller.

- If the program runs on an external controller, you can configure whether the default controller is the external controller or the remote controller with the lowest numbered logical address. With this behavior, if you write a program on an embedded controller referring to the controller as logical address –1, you can swap the embedded controller configuration with an external controller configuration without changing your source code.

Notice that –1 is never a valid value for the **extender** parameter. In addition, the logical addresses of embedded and external controllers also are never valid values for the **extender** parameter. The **extender** parameter refers only to devices that can map interrupt lines, trigger lines, or other signals into or out of a frame.

Error Handling Parameter

Each of the NI-VXI VIs contain error in and error out clusters that keep track of error information as the VIs execute. If the error in cluster passed to a VI already contains an error the VI will not execute. Instead, it only passes on the same error information. Otherwise, it executes and passes the status of its execution out the error out terminal. For more information about the error codes used by the VIs and how they relate to the status codes used by older versions of the NI-VXI VIs, refer to Appendix A, *Error Codes*.

Converting Old Applications

Existing programs automatically load the previous versions of the NI-VXI VIs. These VIs have the word "Old" on the icon and have `_old` added to their names to distinguish them from the new VIs. You can combine old and new VIs when you convert or add on to existing applications. However, only the new VIs use error clusters. The old VIs are located in `\labview\vi.lib\instr\vxi\old`.

2

System Configuration and VXI Library Initialization VIs

This chapter describes the System Configuration VIs. Your application program can use these VIs to copy all of the Resource Manager (RM) table information into data structures at startup so you can find device names or logical addresses by specifying certain attributes of the device for identification purposes.

Locating System Configuration VIs in LabVIEW

Select **Window»Show Diagram** to open the block diagram in LabVIEW. Select **Functions»Instrument I/O»I/O Compatibility»VXI»System Configuration** to locate the System Configuration VIs in LabVIEW.

Finding Help Online for System Configuration VIs

You can find helpful information about individual VIs online by using the LabVIEW Help window. Open the Help window by choosing **Help»Show Context Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You also can double-click the VI to open the front panel. When the Help window is open, you can find more information about each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

Handling Errors

The NI-VXI VIs use the LabVIEW error cluster method for handling errors. Each VI has error in and error out terminals. Refer to Appendix A, *Error Codes*, for information about how the error codes relate to the previous status codes used by the NI-VXI VIs and a description of **error in** and **error out** parameters.

System Configuration VI Descriptions

This section describes System Configuration VIs, which you can use to disable or interrupt dynamic memory, allocate space, and set info for the device information table. You also can find a VXI device with specified attributes in the RM table.

CloseVXIlibrary

Disables, interrupts, and frees dynamic memory allocated for the internal device information table. You should call this VI before exiting your application.
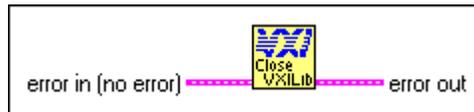
error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

CreateDevInfo

Allocates space in the device information table for a new entry with logical address **la**. The fields in the device information table for the entry are set to default values (null or unasserted values).
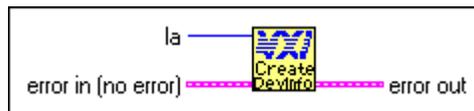
la is the logical address of the device for which to create an entry in the device information table.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information on error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise,

error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

FindDevLA

Finds a VXI device with the specified attributes in the RM table and returns its logical address. You can use any combination of attributes to specify a device. In this manner, you can acquire unknown device names or logical addresses. If **namepat** is "" or any other attribute is −1, or not connected, that attribute is not used in the matching algorithm. If two or more devices match, **la** contains the logical address of the first device found.
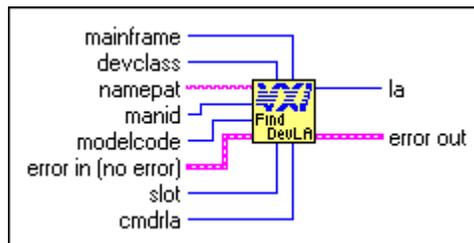
mainframe is the mainframe location of the device (logical address of mainframe extender).

devclass is the device class of the device.
- 0:   Memory Class Device.
- 1:   Extended Class Device.
- 2:   Message-Based Device.
- 3:   Register-Based Device.

namepat is the name pattern. A partial name is acceptable (for example, for GPIB-VXI, it will accept GP).

manid is the VXI manufacturer identification number.

modelcode is the 12-bit or 16-bit model number of the VXI manufacturer.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

slot is the slot location of the device.

cmdrla is the logical address of the Commander.

la is the logical address of the device found.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetDevInfoLong

Obtains information about a specified device from the device information table. The information is contained in a 32-bit unsigned integer.

la is the logical address of the device to get information about.

field is the field identification number.
  12: Base of A24/A32 memory.
  13: Size of A24/A32 memory.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

longvalue is the information for that field.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetDevInfoShort

Obtains information about a specified device from the device information table. The information is contained in a 16-bit unsigned integer.

la is the logical address of the device to get information about.

field is the field identification number.

2: Logical address of the Commander.

3: Mainframe.

4: Slot.

5: Manufacturer identification number.

7: Model code.

9: Device class.

10: Extended subclass (if extended class device).

11: Address space used.

14: Memory type and access time.

15: Bit vector list of VXI interrupter lines.

16: Bit vector list of VXI interrupt handler lines.

17: Mainframe extender, controller information.

Bits	Description
15 and 14	Reserved
13	1: Remote controller 0: Not remote controller
12	1: Child side extender 0: Parent side extender
11	1: Frame extender 0: Not frame extender
10	1: Extended controller
9	1: Embedded controller
8	1: External controller
7 to 0	Frame extender towards root frame

18: Asynchronous mode control state.

19: Response enable state.

20: Protocols supported.

21: Capability/status flags.

22: Status state (Pass/Fail, Ready/Not Ready).

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

shortvalue is the information for that field.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetDevInfoStr

Obtains information about a specified device from the device information table. The information is contained in **stringvalue**.
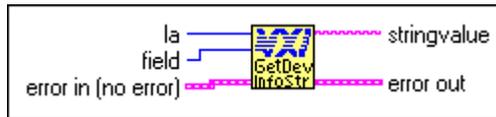
la is the logical address of the device to get information about.

field is the field identification number.
- 1:   Device name.
- 6:   Manufacturer name.
- 8:   Model name.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

**stringvalue** is the information for that field.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

InitVXIlibrary

Allocates and initializes the data structures required by the VXI library VIs in the driver. This VI reads the RM table file and copies all of the RM information into data structures in local memory. It also performs other initialization operations, such as installing the default interrupt handlers.
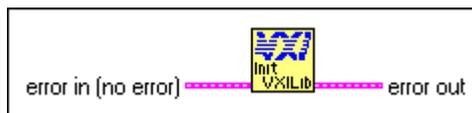
error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetDevInfoLong

Sets information about a specified device in the device information table. The value being set is a 32-bit unsigned integer.

la is the logical address of the device to set information for.

field is the field identification number.
 12: Base of A24/A32 memory.
 13: Size of A24/A32 memory.

longvalue is the information for that field.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetDevInfoShort

Sets information about a specified device in the device information table. The value being set is a 16-bit unsigned integer.

la is the logical address of the device to set information for.

field is the field identification number.

    2: Logical address of the Commander.
    3: Mainframe.
    4: Slot.
    5: Manufacturer identification number.
    7: Model code.
    9: Device class.
  10: Extended subclass (if extended class device).
  11: Address space used.
  14: Memory type and access time.
  15: Bit vector list of VXI interrupter lines.
  16: Bit vector list of VXI interrupt handler lines.
  17: Mainframe extender, controller information.

Bits	Description
15 and 14	Reserved
13	1: Remote controller 0: Not remote controller
12	1: Child side extender 0: Parent side extender
11	1: Frame extender 0: Not frame extender
10	1: Extended controller
9	1: Embedded controller
15 to 13	Reserved
12	1: Child side extender 0: Parent side extender

  18: Asynchronous mode control state.
  19: Response enable state.

20:   Protocols supported.
21:   Capability/status flags.
22:   Status state (Pass/Fail, Ready/Not Ready).

shortvalue is the information for that field.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetDevInfoStr

Sets information about a specified device in the device information table. The information being set is contained in a string.
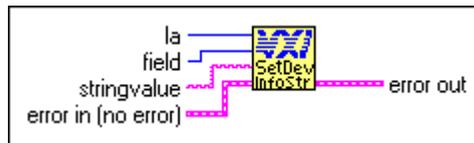
la is the logical address of the device to set information for.

field is the field identification number.
   1:   Device name.
   6:   Manufacturer name.
   8:   Model name.

stringvalue is the buffer to receive information for that field.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

**error out** contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

3

Word Serial Commander Protocol VIs

This chapter describes the VXI Word Serial Commander Protocol VIs. Word Serial communication is the minimal mode of communication between VXI Message-Based devices within the VXI Commander/Servant hierarchy.

Locating Word Serial Commander VIs in LabVIEW

Select **Window»Show Diagram** to open the block diagram in LabVIEW. Select **Functions»Instrument I/O»I/O Compatibility»VXI»Word Serial Commander Protocol** to locate the Word Serial Commander Protocol VIs in LabVIEW.

Finding Help Online for Word Serial Commander VIs

You can find helpful information about individual VIs online by using the LabVIEW Help window. Open the Help window by choosing **Help»Show Context Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You also can double-click the VI to open the front panel. When the Help window is open, you can find more information about each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

Handling Errors

The NI-VXI VIs use the LabVIEW error cluster method for handling errors. Each VI has error in and error out terminals. Refer to Appendix A, *Error Codes*, for information about how the error codes relate to the previous status codes used by the NI-VXI VIs and a description of **error in** and **error out** parameters.

Word Serial Commander VI Descriptions

This section describes World Serial Commander VIs, which you can use to abort an operation in progress and send a command or query to a Message-Based device.

WSabort

Performs a Forced or Unrecognized (Unsupported) Command abort of a Word Serial operation(s) in progress.
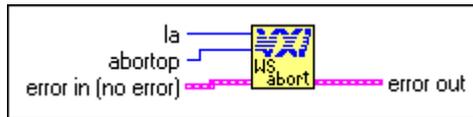
la is the logical address of the Message-Based device.

abortop is the operation to abort.
- 1: Forced Abort: aborts WSwrt, WSrd, and WStrg.
- 2: UnSupCom: aborts WScmd, WSLcmd, and WSEcmd.
- 3: Forced Abort: aborts WScmd, WSLcmd, and WSEcmd.
- 4: Forced Abort: aborts all Word Serial operations.
- 5: Async Abort: aborts all Word Serial operations immediately. Be careful when using this option. During a Word Serial query, the Servant may be left in an invalid state if the operation is aborted after writing the query and before reading the response register. When using this option, the Word Serial operation is aborted immediately as compared to using options 1, 3, and 4, where the operation is not aborted until the response is read in that situation.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSclr

Sends the Word Serial Clear command to a Message-Based device.

**la** is the logical address of the Message-Based device.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WScmd

Sends a Word Serial command or query to a Message-Based device.

la is the logical address of the Message-Based device.

cmd is the Word Serial command value.

respflag is a Boolean value.
    TRUE: Get a response (query).
    FALSE: Do not get a response.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

response is the 16-bit response.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise,

error out describes the error status of this VI. Refer to Appendix A,
Error Codes, for specific error information.

WSEcmd

Sends an Extended Longword Serial command or query to a Message-Based device.
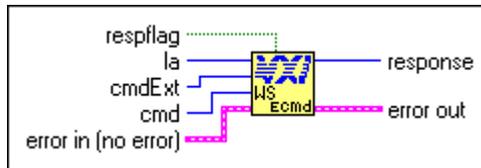
respflag is a Boolean value.
>   TRUE: Get a response (query).
>   FALSE: Do not get a response.

la is the logical address of the Message-Based device.

cmdExt is the upper 16 bits of the 48-bit Extended Longword Serial
command value.

**cmd** is the lower 32 bits of the 48-bit Extended Longword Serial command
value.

error in describes error conditions occurring before the VI executes. Refer
to Appendix A, *Error Codes*, for more information about error clusters and
specific errors.

response is the 32-bit location to store the response.

error out contains error information. If the error in cluster indicated an
error, the **error out** cluster contains the same information. Otherwise,
error out describes the error status of this VI. Refer to Appendix A,
Error Codes, for specific error information.

WSgetTmo

Obtains the actual time period to wait before aborting a Word Serial, Longword Serial, or
Extended Longword Serial Protocol transfer.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

actualtimo is the timeout period (in milliseconds).

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSLcmd

Sends a Longword Serial command or query to a Message-Based device.
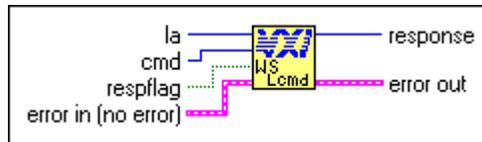
la is the logical address of the Message-Based device.

cmd is the Longword Serial command value.

respflag is a Boolean value.
    TRUE: Get a response (query).
    FALSE: Do not get a response.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

response is the 32-bit location to store the response.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSLresp

Retrieves a response to a previously sent Longword Serial Protocol query from a VXI Message-Based device.

WSLcmd can send a query and automatically read a response. However, if you must break up the sending of the query and the reading of the response, you can use WSLcmd to send the query without reading the response and WSLresp to read the response.
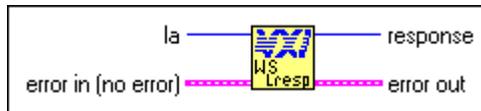
Note This VI is intended for debugging purposes only.

la is the logical address of the Message-Based device.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

response is the 32-bit response.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSrd

Transfers the specified number of data bytes from a Message-Based device into a specified local memory buffer, using the VXIbus Byte Transfer Protocol.
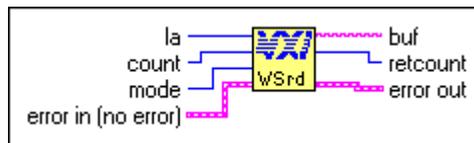
la is the logical address of the Message-Based device from which the buffer is read.

count is the maximum number of bytes to transfer.

U16

mode is the transfer mode bit vector. The following table describes the mode bit vector corresponding to bits 15 through 0.

Bit	Event Signal
15 to 8	EOS character (valid if EOS termination)
4	EOS character termination 1: Terminate transfer on EOS bit. 0: Do not terminate transfer on EOS bit.
3	CR character termination 1: Terminate transfer on CR bit. 0: Do not terminate transfer on CR bit
2	LF character termination 1: Terminate transfer on LF bit. 0: Do not terminate transfer on LF bit
1	END bit termination suppression 0: Terminate transfer on END bit. 1: Do not terminate transfer on END bit.
0	Not DOR 0: Abort if not DOR. 1: Poll until DOR

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

buf is the data read.

retcount is the number of bytes actually transferred.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSresp

Retrieves a response for a previously sent Word Serial Protocol query from a VXI Message-Based device.

The WScmd VI can send a query and automatically read a response. However, if it is necessary to break up the sending of the query and the reading of the response, you can use the WScmd VI to send the query without reading the response and use the WSresp VI to read the response.

Note   This VI is intended for debugging purposes only.

la is the logical address of the Message-Based device.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

response is the 16-bit response.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSsetTmo

Sets the time period to wait before aborting a Word Serial, Longword Serial, or Extended Longword Serial Protocol transfer. It returns the actual timeout value set (the nearest timeout period possible greater than or equal to the timeout period specified).
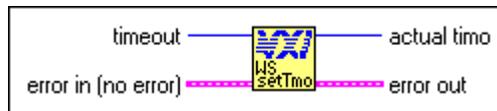
timeout is the timeout period (in milliseconds).

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

actual timo is the actual timeout period set (in milliseconds).

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WStrg

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

la is the logical address of the Message-Based device.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSwrt

Transfers the specified number of data bytes from a memory buffer to a Message-Based device, using the VXIbus Byte Transfer Protocol.
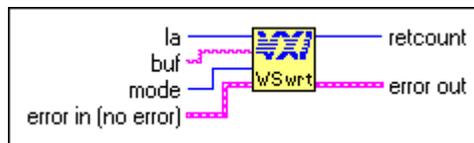
la is the logical address of the Message-Based device to which the buffer is written.

buf is the write buffer.

mode is the transfer mode bit vector,

    for Bit 0

        1:   Poll until device is DIR.

        0:   Abort if device is not DIR.

    for Bit 1

        1:   Set the END bit on the last byte of the transfer.

        0:   Clear the END bit on the last byte of the transfer.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

**retcount** is the number of bytes actually transferred.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

4

Word Serial Servant
Protocol VIs

This chapter describes the VXI Word Serial Servant Protocol VIs. Word Serial communication is the minimal mode of communication between VXI Message-Based devices within the VXI Commander/Servant hierarchy. The local CPU (the CPU on which the NI-VXI functions are running) uses the Word Serial Servant VIs to perform VXI Message-Based Word Serial Servant communication with its Commander.

You use these VIs only in the case where the local CPU is not a Top-Level Commander (probably not the Resource Manager), such as in a multiple CPU situation. In a multiple CPU situation, the local CPU must allow the Resource Manager device to configure the local CPU and can optionally implement some basic message-transfer Word Serial communication with its Commander. The four basic types of Word Serial Servant VIs are as follows:

- Command reception

- Query reception and responding

- Buffer sending

- Buffer receiving

Word Serial Protocol is a simple 16-bit transfer protocol between a Commander and its Servants. The Commander polls specific bits in the VXI Response register of the Servant to determine when a command can be written, when a response can be read from the Data Low register, and when a Word Serial protocol error occurs. Before a Commander can send a Word Serial command to a Servant, it must first poll the Write Ready (WR) bit until it is asserted (set to 1). The Commander can then write the command to the Data Low register.

If the Commander is sending a query, it first sends the query in the same manner as sending a command, but then continues by polling the Read Ready (RR) bit until it is asserted. It then reads the response from the Data Low register. A buffer write is a series of Byte Available Word Serial commands sent to the Servant, with the additional constraint that the Data In Ready (DIR) bit as well as the WR bit must be asserted before the

Byte Available is sent. The lower 8 bits (bits 0 to 7) of the 16-bit command contains a single byte of data (bit 8 is the END bit). Therefore, one *Byte Available* is sent for each data byte in the buffer written.

A buffer read is a series of *Byte Request* Word Serial queries sent to the Servant, with the additional constraint that the Data Out Ready (DOR) bit, as well as the WR bit, must be asserted before the *Byte Request* is sent. The lower 8 bits (bits 0 to 7) of the 16-bit response contain a single byte of data (bit 8 is the END bit). Therefore, one *Byte Request* is sent for each data byte in the buffer read.

In addition to the WR, RR, DIR, and DOR bits that get polled during various Word Serial transfers, the ERR* bit is also checked. The ERR* bit indicates that a Word Serial Protocol error has occurred. The Word Serial Protocol error can be Unsupported Command, Multiple Query Error (MQE), DIR Violation, DOR Violation, RR Violation, or WR Violation. The Word Serial Servant Protocol VIs allow the local CPU to generate any of the Word Serial Protocol errors and to respond to the *Read Protocol Error* Word Serial query with the corresponding protocol error. The ERR* bit assertion and unassertion are handled automatically.

The Longword Serial and Extended Longword Serial Protocols are similar to the Word Serial Protocol, but involve 32-bit and 48-bit command transfers, respectively, instead of the 16-bit transfers of the Word Serial Protocol. The VXI specification, however, provides no common command usages for these protocols. The commands are either VXI Reserved or User-Defined. The NI-VXI interface gives you the ability to receive and process any one of these commands.

Locating Word Serial Servant Protocol VIs in LabVIEW

Select **Window»Show Diagram** to open the block diagram in LabVIEW. Select **Functions»Instrument I/O»I/O Compatibility»VXI»Word Serial Servant** to locate the Word Serial Servant Protocol VIs in LabVIEW.

Finding Help Online
for Word Serial Servant Protocol VIs

You can find helpful information about individual VIs online by using the LabVIEW Help window. Open the Help window by choosing **Help»Show Context Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You can also double-click the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

Handling Errors

The NI-VXI VIs use the LabVIEW error cluster method for handling errors. Each VI has error in and error out terminals. Refer to Appendix A, *Error Codes*, for information about how the error codes relate to the previous status codes used by the NI-VXI VIs and a description of **error in** and **error out** parameters.

Word Serial Servant VI Descriptions

This section describes World Serial Servant VIs, which you can use to generate errors, return the address of the WSScmd interrupt handler, respond to the *Read Protocol Error* query, and replace the current WSScmd interrupt handler.

GenProtError

Generates a Word Serial protocol error if one is not already pending. The Response register bit ERR* is asserted if the value of the protocol error, **proterr**, is not FFFF. If **proterr** is FFFF, ERR* is unasserted. If no previous error existed, the **proterr** value is saved for response to a future *Read Protocol Error* query via the VI RespProtError.
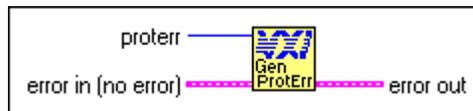
proterr is the protocol error to generate an error.
    FFFF: Clear any protocol error condition.
    FFFD: Multiple Query Error (MQE).

FFFC: Unsupported Command (UnSupCom).
FFFB: Data In Ready violation (DIRviol).
FFFA: Data Out Ready violation (DORviol).
FFF9:   Read Ready violation (RRviol).
FFF8:   Write Ready violation (WRviol).

Other values are reserved.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetWSScmdHandler

Returns the address of the current WSScmd interrupt handler.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

func is the address of the current WSScmd interrupt handler.
    0:    Word Serial Servant VIs not supported.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetWSSEcmdHandler

Returns the address of the current WSSEcmd interrupt handler.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

func is the address of the current WSSEcmd interrupt handler.
  0:    Word Serial Servant VIs not supported.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetWSSLcmdHandler

Returns the address of the current WSSLcmd interrupt handler.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

func is the address of the current WSSLcmd interrupt handler.
  0:    Word Serial Servant VIs not supported.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetWSSrdHandler

Returns the address of the current WSSrd done notification interrupt handler.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

func is the address of the current WSSrd done notification interrupt handler.
    0:   Word Serial Servant VIs not supported.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetWSSwrtHandler

Returns the address of the current WSSwrt done notification interrupt handler.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

func is the address of the current WSSwrt done notification interrupt handler.
    0:   Word Serial Servant VIs not supported.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

RespProtError

Responds to the Word Serial *Read Protocol Error* query with the last protocol error generated via the GenProtError VI. The ERR* bit is unasserted.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetWSScmdHandler

Replaces the current WSScmd interrupt handler with a specified handler.
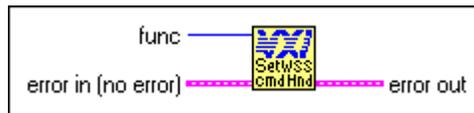
func is the address of the new WSScmd interrupt handler obtained from the GetWSScmdHandler VI.

    0:   Set to DefaultWSScmdHandler.
    3:   LabVIEW Occurrence Handler.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetWSSEcmdHandler

Replaces the current WSSEcmd interrupt handler with a specified handler.

func is the address of the new WSSEcmd interrupt handler obtained from the GetWSScmdHandler VI.

    0:    Set to DefaultWSSEcmdHandler.

    3:    LabVIEW Occurrence Handler.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetWSSLcmdHandler

Replaces the current WSSLcmd interrupt handler with a specified handler.
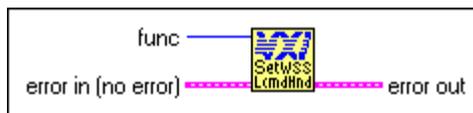
func is the address of the new WSSLcmd interrupt handler obtained from the GetWSScmdHandler VI.

    0:    Set to DefaultWSSLcmdHandler.

    3:    LabVIEW Occurrence Handler.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer Appendix A, *Error Codes*, for specific error information.

SetWSSrdHandler

Replaces the current WSSrd done notification interrupt handler with a specified handler.

func is the address of the new WSSrd done notification handler obtained from the GetWSScmdHandler VI.

    0:    Set to DefaultWSSrdHandler.
    3:    LabVIEW Occurrence Handler.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetWSSwrtHandler

Replaces the current WSSwrt done notification interrupt handler with a specified handler.
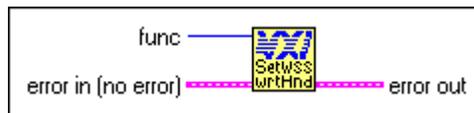
func is the address of the new WSSwrt done notification handler obtained from the GetWSScmdHandler VI.

    0:    Set to DefaultWSSwrtHandler.
    3:    LabVIEW Occurrence Handler.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSSabort

Aborts the Word Serial Servant operation(s) in progress.

I16

abortop is the operation to abort, bit vector.

Bit	Description
0	Abort WSSwrt
1	Abort WSSrd
2	Abort WSSsendResp
15	Initialize Word Serial Servant hardware. This includes aborting all Word Serial operations, clearing out errors, removing all pending Word Serial Servant interrupts, and disabling the interrupts.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSScmdHandler

Waits until a Word Serial Protocol command or query is received from a VXI Message-Based Commander.
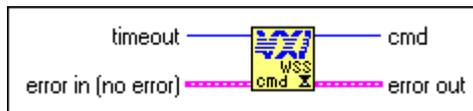
I32

timeout specifies the number of milliseconds to wait for the interrupt.
    –1: Forever.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

cmd is the 16-bit Word Serial command received.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSSdisable

Desensitizes the local CPU to interrupts generated when a Word Serial command is written to the Data Low register or when a response is read from the Data Low register.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSSEcmdHandler

Waits until an Extended Longword Serial Protocol command or query is received from a VXI Message-Based Commander.
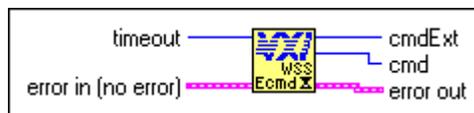
timeout specifies the number of milliseconds to wait for the interrupt.
   –1: Forever.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

cmdExt is the upper 16 bits of the 48-bit Extended Longword Serial command value.

cmd is the lower 32 bits of the 48-bit Extended Longword Serial command value.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSSenable

Sensitizes the local CPU to interrupts generated when a Word Serial command is written to the Data Low register or when a response is read from the Data Low register.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSSLcmdHandler

Waits until a Longword Serial Protocol command or query is received from a VXI Message-Based Commander.
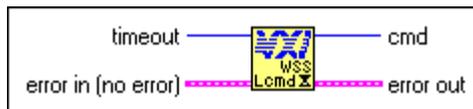
timeout specifies the number of milliseconds to wait for the interrupt.

–1: Forever.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

cmd is the 32-bit Longword Serial command received.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSSLnoResp

Acknowledges a received Longword Serial Protocol command that has no response and asserts the Write Ready (WR) bit in the local CPU Response register. You must call this VI after the processing of a Longword Serial Protocol command (queries are responded to with WSSLsendResp).
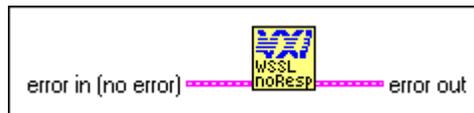
error in (no error) ———— error out

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSSLsendResp

Responds to a received Longword Serial Protocol query with a response and asserts the WR bit in the local CPU Response register. You must call this VI after the processing of a Longword Serial Protocol query (commands are acknowledged with the WSSLnoResp VI).

response ————
error in (no error) ———— error out
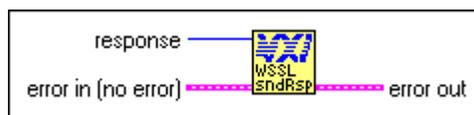
response is a 32-bit response.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSSnoResp

Acknowledges a received Word Serial Protocol command that has no response and asserts the WR bit in the local CPU Response register. You must call this VI after the processing of a Word Serial Protocol command (queries are responded to with the WSSsendResp VI).


error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSSrd

Posts a read operation to begin receiving the specified number of data bytes from a Message-Based Commander into a specified memory buffer, using the VXIbus Byte Transfer Protocol.
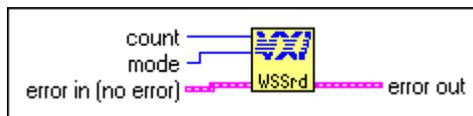
count is the maximum number of bytes to transfer.

mode is the transfer mode bit vector.

Bit	Description
0	Determines the DIR signal mode to Commander 0: Do not send DIR signal to Commander. 1: Send DIR signal to Commander.
1-15	0: Reserved.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSSrdHandler

Waits until a Word Serial Servant read operation (started with the WSSrd VI) terminates.
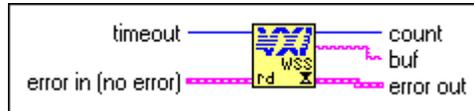
timeout specifies the number of milliseconds to wait for the interrupt.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

count is the actual number of bytes received.

buf is the buffer received from WSSrd operation.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSSsendResp

Responds to a received Word Serial Protocol query with a response and asserts the WR bit in the local CPU Response register. You must call this VI after the processing of a Word Serial Protocol query (commands are acknowledged with the WSSnoResp VI).
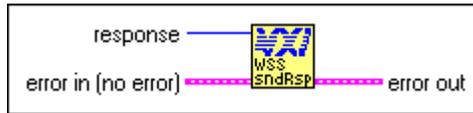
response is the 16-bit response.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSSwrt

Posts the write operation to transfer the specified number of data bytes from a specified memory buffer to the Message-Based Commander, using the VXIbus Byte Transfer Protocol.
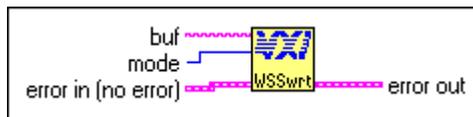
buf is the write buffer.

mode is the mode of transfer (bit vector).

Bit	Description
0	Determines the DOR signal mode to Commander (if enabled). 0: Do not send DOR signal to Commander. 1: Send DOR signal to Commander.

Bit	Description
1	Specifies the END bit termination with last byte. 0: Do not END with the last byte. 1: Send END with the last byte.
2 to 15	0: Reserved.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WSSwrtHandler

Waits until a Word Serial Servant write operation (started with the WSSwrt VI) terminates.
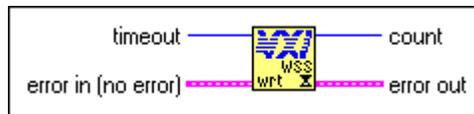
timeout specifies the number of milliseconds to wait for the interrupt.
  –1: Forever.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

count is the actual number of bytes sent.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

5

Low-Level VXIbus Access VIs

This chapter describes how to use the VIs that give you the fastest access method for directly reading from or writing to any of the VXIbus address spaces.

There are several situations in which you must direct reads and writes to the different VXIbus address spaces, including some of the following:

- Register-Based device/instrument drivers
- Non-VXI/VME device/instrument drivers
- Accessing device-dependent registers on any type of VXI/VME device
- Implementing shared memory protocols

Low-level and high-level access to the VXIbus, as the NI-VXI interface defines them, are very similar in nature. Both sets of VIs can perform direct reads from and writes to any VXIbus address space with any privilege state or byte order. However, the two interfaces have different emphases with respect to user protection, error checking, and access speed.

Low-level VXIbus access is the fastest way to directly read from or write to the VXIbus address spaces. You access VXIbus address spaces by reading and writing to offsets in the local CPU address space that correspond to addresses on the VXIbus.

The address space of the local CPU is mapped onto the VXIbus in areas called windows. The size and number of windows varies, depending on the hardware. The size of the window is always a power of two, where a multiple of the size of the window would encompass an entire VXIbus address space. The window base register determines the multiple that a window can currently access.

National Instruments MITE-based controllers, such as the PCI-MXI-2 and the 700 and 800 series embedded controllers, allow you to configure the size of the address space reserved for windows into VXIbus address spaces.

Non-MITE based controllers have a fixed number of windows that each have a fixed size. The documentation that came with your hardware describes these specifications. When using non-MITE based controllers,

the size parameter to the MapVXIAddress VI is ignored. The size parameter is supported only on Windows 32-bit and Solaris 2 operating systems.

✎ **Note**   You typically have the same access privileges and byte orders for all devices. The VXIbus specification requires that VXI devices respond to the supervisory data privilege state (address modifier codes). This increases the overall throughput of the program. Otherwise, the application must continually restore the state of the windows into VXIbus address spaces.

NI-VXI uses a term within this chapter called the *hardware context* (or window). The hardware context for a VXI window consists of the VXI address space being accessed, the base offset into the address space, the access privilege, and the byte order for the accesses through the window. Before accessing a particular address, you must set up the window with the appropriate hardware context using the MapVXIAddress VI. This VI returns an address pointer you can use for accessing the window in the future by using the VXIpeek and VXIpoke VIs.

Multiple Accessors for a Window

Sometimes problems occur when an application requires different privilege states, byte orders, and/or base addresses within the same window. If the hardware context is changed by a subsequent call to MapVXIAddress or by other VIs such as SetPrivilege or SetByteOrder, previously mapped windows would not have their intended access parameters. There are two types of access privileges to a window that help solve this problem: *Owner Privilege* and *Access Only Privilege*. These two privileges define which caller of the MapVXIAddress VI can change the settings of the corresponding window.

Owner Privilege

A caller can obtain Owner Privilege to a window by requesting owner privilege in the MapVXIAddress VI (via the **accessparms** parameter). This address mapping will not succeed if another address pointer with Owner Privilege or Access Only Privilege has already been mapped for that window. If the mapping succeeds, the VI returns a valid pointer and a non-negative status value. The **window** output parameter returned from the MapVXIAddress VI associates the address pointer returned from the VI with a particular window and also signifies Owner Privilege to that window. Owner Privilege access is complete and exclusive. The caller can use the SetPrivilege, SetByteOrder, and SetContext VIs with this window

to dynamically change the access privileges. Notice that if the execution of the MapVXIAddress VI succeeds for either Owner Privilege or Access Only Privilege, the pointer remains valid in both cases until the UnMapVXIAddress VI is executed for the corresponding window. The advantage of Owner Privilege access is that it gives complete and exclusive access for that window to the caller, so you can dynamically change the access privileges. Because no other callers can succeed, there is no problem with destroying the access state of another caller.

Access Only Privilege

You can obtain Access Only Privilege for a window by requesting access only privileges in the MapVXIAddress VI. With this privilege mode, you can simultaneously have multiple address pointers to access a particular window, while still guaranteeing that the hardware context does not change between accesses. The VI executes successfully under either of the following conditions:

*   No address pointers are mapped for the window (first caller for Access Only Privilege for this window). The hardware context is set as requested in the call. The call returns a successful status and a valid address pointer and **window** for Access Only Privilege.

*   No address pointer has been mapped with Owner Privilege for the required window. There *are* address pointers with Access Only Privilege, but they are using the same hardware context (privilege state, byte order, address range) for their accesses to the window. Because the hardware context is compatible, it does not need to be changed. The VI returns a successful status and a valid address pointer and **window** for Access Only Privilege.

The successful call returns a valid pointer and a non-negative return value. The 32-bit window number signifies that the access privileges to the window are Access Only Privilege.

With Access Only Privilege, you cannot use the SetPrivilege, SetByteOrder, and SetContext VIs in your application to dynamically change the hardware context. No Access Only accessor can change the state of the window. The initial Access Only call sets the hardware context for the window, which cannot be changed until all Access Only accessors have called the UnMapVXIAddress VI to free the window.

The GetPrivilege, GetByteOrder, and GetContext VIs succeed regardless of whether the caller has Owner Privilege or Access Only Privilege.

Locating Low-Level VXIbus Access VIs in LabVIEW

Select **Window»Show Diagram** to open the block diagram in LabVIEW. Select **Functions»Instrument I/O»I/O Compatibility»VXI» Low-Level VXIbus Access** to locate the Low-Level VXIbus Access VIs in LabVIEW.

Finding Help Online for Low-Level VXIbus Access VIs

You can find helpful information about individual VIs online by using the LabVIEW Help window. Open the Help window by choosing **Help»Show Context Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You can also double-click the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

Handling Errors

The NI-VXI VIs use the LabVIEW error cluster method for handling errors. Each VI has error in and error out terminals. Refer to Appendix A, *Error Codes*, for information about how the error codes relate to the previous status codes used by the NI-VXI VIs and a description of **error in** and **error out** parameters.

Low-Level VXIbus Access VI Descriptions

This section describes Low-Level VXIbus Access VIs, which you can use to obtain byte/word order of data, the current hardware interface settings, and the current VXI/VME access privilege for the specified window. You also can use these VIs to return information about the state of the VXIbus for a field in a controller.

GetByteOrder

Gets the byte/word order of data transferred into or out of the specified window.

window, as obtained from MapVXIAddress.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

ordermode is the byte/word order of data.
  0:   Motorola byte ordering.
  1:   Intel byte ordering.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetContext

Gets the current hardware interface settings (context) for the specified window.
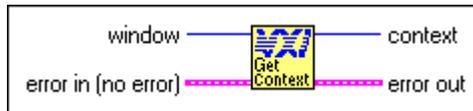
window, as obtained from MapVXIAddress.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

context is the VXI hardware access context.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetPrivilege

Gets the current VXI/VME access privilege for the specified window.
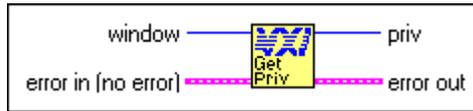
| U32 | **window**, as returned from MapVXIAddress. |

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

priv is the 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.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetVXIbusStatus

Returns information about the state of the VXIbus in a specified controller (either an embedded CPU or an extended controller).
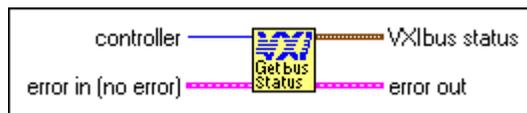
controller is the controller from which to get the status (–2: OR of all).

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

VXIbus status is a cluster containing the VXIbus status, being comprised of the following elements:

I16   **Bus Error** is where a value of 1 means that a bus error occurred on the last access.

I16   **Sysfail** is where a value of 1 means that SYSFAIL* is asserted.

I16   **ACfail** is where a value of 1 means that ACFAIL* is asserted.

I16   **Signal In** is the number of signals queued.

I16   **VXI ints** is a bit vector, where a value of 1 in bit positions 0 through 6 means that the interrupt 1 through 7 is asserted.

I16   **ECL trigs** is a bit vector, where a value of 1 in bit positions 0 through 5 means that the trigger 0 through 5 is asserted.

I16   **TTL trigs** is a bit vector, where a value of 1 in bit positions 0 through 7 means that the trigger 0 through 7 is asserted.

A value of –1 returned in any of the fields of the cluster signifies that there is no hardware support to retrieve information for that particular VXIbus state.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetVXIbusStatusInd

Returns information about the state of the VXIbus for the specified field in a particular controller.

```
controller ───────┐  ┌──────── VXIbus status
        field ─────┤  │ Get bus
error in (no error) ═══│ Stat Ind├═══ error out
```

I16   **controller** is the controller from which to get the status (–2: OR of all).

U16   **field** is the number of the field about which to return information.
- 1: **Bus Error**, where a value of 1 means that the last access BERRed.
- 2: **Sysfail**, where a value of 1 means that SYSFAIL* is asserted.
- 3: **ACfail**, where a value of 1 means that ACFAIL* is asserted.
- 4: **Signal In**, which is the number of signals queued.
- 5: **VXI ints**, which is a bit vector, where a value of 1 in bit positions 0 through 6 means that the interrupt 1 through 7 is asserted.

6:   **ECL trigs**, which is a bit vector, where a value of 1 in bit positions 0 through 5 means that the trigger 0 through 5 is asserted.

7:   **TTL trigs**, which is a bit vector, where a value of 1 in bit positions 0 through 7 means that the trigger 0 through 7 is asserted.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

VXIbus status in which a value of –1 in any of the fields means that there is no hardware support for that particular state.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetWindowRange

Gets the range of addresses that a particular window, allocated with the MapVXIAddress VI, can currently access within a particular VXIbus address space.

window, as obtained from MapVXIAddress.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

windowbase is the base VXI address.

windowend is the end VXI address.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

MapVXIAddress

Sets up a window into one of the VXI address spaces according to the access parameters specified, and returns a pointer to a local CPU address that accesses the specified VXI address. This VI also returns the window ID associated with the window, which is used with all other low-level VXIbus access VIs.
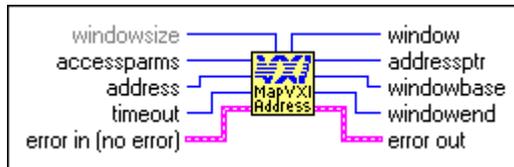
`U32`   **windowsize** is the size of the window to be mapped (default is 64K).

Note   The size parameter is only supported on Win32 and Solaris 2 platforms using MITE-based hardware.

`U16`   **accessparms** specifies the access parameters.

Bits 0 and 1 are used to specify the VXI address space.

　　1: A16.
　　2: A24.
　　3: A32.

Bits 2 through 4 are used to specify the 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.

Bit 5 = 0.

Bit 6 is used to specify the access mode.

　　0: Access only.
　　1: Owner access.

Bit 7 is used to specify the byte order.

　　0: Motorola.
　　1: Intel.

Bits 8 through 15 = 0.

`U32`   **address** is the address within A16, A24, or A32.

`I32`   **timeout** is the timeout (in milliseconds).

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

window is the window number for use with other VIs.

addressptr is the pointer to local address for specified VXI address.
> 0:   Unable to get pointer.

Note   To maintain compatibility and portability, use the pointer returned by this VI only with the VXIpeek and VXIpoke VIs.

windowbase is the base VXI address.

windowend is the end VXI address.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetByteOrder

Sets the byte/word order of data transferred into or out of the specified window.

window, as obtained from MapVXIAddress.

ordermode is the byte/word order of data.
> 0:   Motorola byte ordering.
> 1:   Intel byte ordering.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetContext

Sets the current hardware interface settings (context) for the specified window. The value for **context** should have been set previously by the GetContext VI.
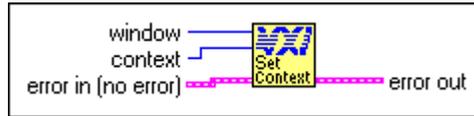
U32	**window**, as returned from MapVXIAddress.
U32	**context** is the VXI hardware access context to install, as returned from GetContext.
[error]	**error in** describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.
[error]	**error out** contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetPrivilege

Sets the VXI/VME access privilege for the specified window to the specified privilege state.
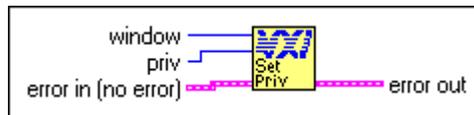
window, as returned from MapVXIAddress.

priv is the 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.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

UnMapVXIAddress

Deallocates a window that was allocated using the MapVXIAddress VI.
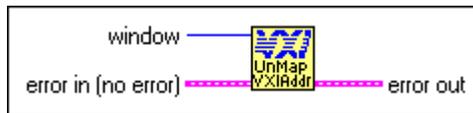
window, as returned from MapVXIAddress.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

**error out** contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

VXIpeek

Reads a single byte, word, or longword from a specified VXI address by de-referencing a pointer obtained from MapVXIAddress.
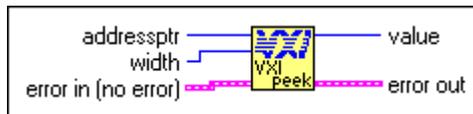
addressptr is the address pointer obtained from MapVXIAddress.

width specifies byte, word, or longword.
- 1: Byte.
- 2: Word.
- 4: Longword.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

value is the data value read.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

VXIpoke

Writes a single byte, word, or longword to a specified VXI address by dereferencing a pointer obtained from MapVXIAddress.
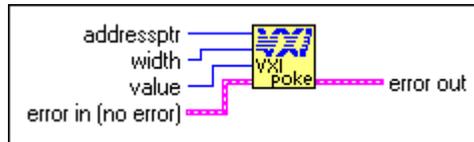
addressptr is the address pointer obtained from MapVXIAddress.

width specifies byte, word, or longword.
- 1:   Byte.
- 2:   Word.
- 4:   Longword.

value is the data value to write.

**error in** describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

6

High-Level VXIbus Access VIs

This chapter describes the VIs with which you have direct access to the VXIbus address spaces. You can use these VIs to read, write, and move blocks of data between any of the VXIbus address spaces. Use these easy-to-use VIs when execution speed is not a critical issue.

Use low-level and high-level VXIbus Access VIs to directly read or write to VXIbus addresses. There are several situations that require you to direct reads and writes to the different VXIbus address spaces, including the following:

- Using register-based device/instrument drivers

- Using non-VXI/VME device/instrument drivers

- Accessing device-dependent registers on any type of VXI/VME device

- Implementing shared memory protocols

Low-level and high-level access to the VXIbus, as the NI-VXI interface defines them, are very similar in nature. Both sets of VIs can perform direct reads of and writes to any VXIbus address space with any privilege state or byte order. However, the two interfaces have different emphases with respect to user protection, error checking, and access speed.

High-level VXIbus access VIs need not take into account any of the considerations that are required by the low-level VXIbus access VIs. The high-level VXIbus access VIs have all the necessary information for accessing a particular VXIbus address wholly contained within the VI input parameters. The parameters prescribe the address space, privilege state, byte order, and offset within the address space. Bus errors are automatically trapped, and an appropriate error status is returned.

More overhead is involved with the use of the high-level VXIbus access VIs, but if overall throughput of a particular access (for example, configuration or small number of accesses) is not the primary concern, the high-level VXIbus access VIs act as an easy-to-use interface that can do any VXIbus accesses necessary for an application.

All accesses to the VXIbus address spaces performed by use of the high-level VXIbus access VIs are fully protected. The hardware interface settings (*context*) for the applicable window are saved on entry to the VI and restored upon exit. No other VIs in the NI-VXI interface, including the low-level VXIbus access VIs, will conflict with the high-level VXIbus access VIs. You can use high-level and low-level VXIbus access VIs at the same time.

Locating High-Level VXIbus Access VIs in LabVIEW

Select **Window»Show Diagram** to open the block diagram in LabVIEW. Select **Functions»Instrument I/O»I/O Compatibility»VXI»High-Level VXIbus Access** to locate the High-Level VXIbus Access VIs in LabVIEW.

Finding Help Online for High-Level VXIbus Access VIs

You can find helpful information about individual VIs online by using the LabVIEW Help window. Open the Help window by choosing **Help»Show Context Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You also can double-click the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

Handling Errors

The NI-VXI VIs use the LabVIEW error cluster method for handling errors. Each VI has error in and error out terminals. Refer to Appendix A, *Error Codes*, for information about how the error codes relate to the previous status codes used by the NI-VXI VIs and a description of **error in** and **error out** parameters.

High-Level VXIbus Access VI Descriptions

This section describes the High-Level VXIbus Access VIs, which you can use to read or write a single byte or word from a specified VXI address and copy a block of memory from a source location.

VXIin

Reads a single byte, word, or longword from a specified VXI address with the specified byte order and privilege state.

U16
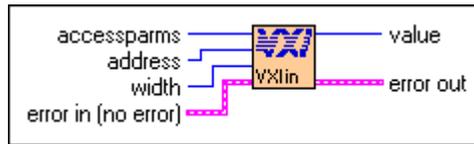
accessparms specifies the access parameters.
    Bits 0 and 1 are used to specify the VXI address space.
        1:   A16.
        2:   A24.
        3:   A32.
    Bits 2 through 4 specify the access privilege.
        0:   Nonprivileged data access.
        1:   Supervisory data access.
        2:   Nonprivileged program access.
        3:   Supervisory program access.
        4:   Nonprivileged block access.
        5:   Supervisory block access.
    Bits 5 and 6 are reserved (should be 0).
    Bit 7 specifies the byte order.
        0:   Motorola.
        1:   Intel.
    Bits 8 through 15 are reserved (should be 0).

U32

address is the VXI address within the specified space.

U16

width is the read width.
    1:   Byte.
    2:   Word.
    4:   Longword.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

value is the value read.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

VXIinReg

Reads a single word from a specified VXI register offset on the specified VXI device. The register is read in Motorola byte order as nonprivileged data.
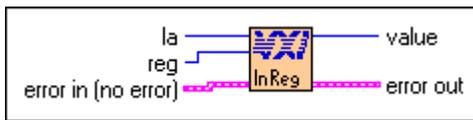
la is the logical address of the device to read from.

reg is the offset within VXI logical address registers.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

value is the value read from the VXI register of the device.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

VXImove

Copies a block of memory from a specified source location in any address space (local, A16, A24, A32) to a specified destination in any address space.
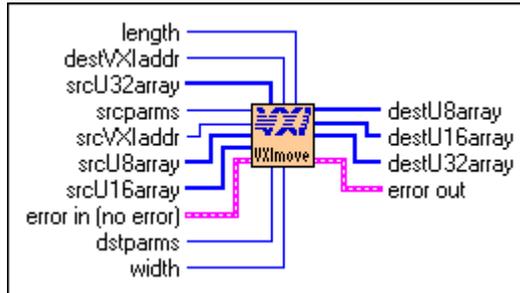
length is the number of elements to transfer.

destVXIaddr is the destination address in the VXI address space (applicable only if the address space specified by **destparms** indicates VXI address space).

srcU32array is the source unsigned longword array in the local address space (applicable only if the address space specified by **srcparms** indicates local address space, and you want to transfer data from an array of longwords).

srcparms specifies the source parameters.
> Bits 0 and 1 are used to specify the source address space.
>> 0:   Local (bits 2, 3, 4, and 7 should be 0).
>> 1:   A16.
>> 2:   A24.
>> 3:   A32.
> Bits 2 through 4 are used to specify the access privilege.
>> 0:   Nonprivileged data access.
>> 1:   Supervisory data access.
>> 2:   Nonprivileged program access.
>> 3:   Supervisory program access.
>> 4:   Nonprivileged block access.
>> 5:   Supervisory block access.
> Bits 5 and 6 are reserved (should be 0).
> Bit 7 is used to specify the byte order.
>> 0:   Motorola.
>> 1:   Intel.
> Bits 8 through 15 are reserved (should be 0).

U32

srcVXIaddr is the source address in the VXI address space (applicable only if the address space specified by **srcparms** indicates VXI address space).

[U8]

srcU8array is the source unsigned byte array in the local address space (applicable only if the address space specified by **srcparms** indicates local address space, and you want to transfer data from an array of bytes).

[U16]

srcU16array is the source unsigned word array in the local address space (applicable only if the address space specified by **srcparms** indicates local address space, and you want to transfer data from an array of words).

[╓╥╖]

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

U16

dstparms specifies the destination parameters.
    Bits 0 and 1 are used to specify the destination address space.
        0:   Local (bits 2, 3, 4, and 7 should be 0).
        1:   A16.
        2:   A24.
        3:   A32.
    Bits 2 through 4 are used to specify the access privilege.
        0:   Nonprivileged data access.
        1:   Supervisory data access.
        2:   Nonprivileged program access.
        3:   Supervisory program access.
        4:   Nonprivileged block access.
        5:   Supervisory block access.
    Bits 5 and 6 are reserved (should be 0).
    Bit 7 is used to specify the byte order.
        0:   Motorola.
        1:   Intel.
    Bits 8 through 15 are reserved (should be 0).

U16

width specifies byte, word, or longword.
    1:   Byte.
    2:   Word.
    4:   Longword.

[U8]

destU8array is the destination unsigned byte array in the local address space (applicable only if the address space specified by **destparms** indicates local address space, and you want to transfer data into an array of bytes).

destU16array is the destination unsigned word array in the local address space (applicable only if the address space specified by **destparms** indicates local address space, and you want to transfer data into an array of words).

destU32array is the destination unsigned longword array in the local address space (applicable only if the address space specified by **destparms** indicates local address space and you want to transfer data into an array of longwords).

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

VXIout

Writes a single byte, word, or longword to a specified VXI address with the specified byte order and privilege state.

value is the data value to write.

accessparms specifies the access parameters.
> Bits 0 and 1 are used to specify the VXI address space.
>> 1:   A16.
>> 2:   A24.
>> 3:   A32.
>
> Bits 2 through 4 are used to specify the access privilege.
>> 0:   Nonprivileged data access.
>> 1:   Supervisory data access.
>> 2:   Nonprivileged program access.
>> 3:   Supervisory program access.
>> 4:   Nonprivileged block access.
>> 5:   Supervisory block access.
>
> Bits 5 and 6 are reserved (should be 0).

Bit 7 specifies the byte order.
    0:    Motorola.
    1:    Intel.
Bits 8 through 15 are reserved (should be 0).

address is the VXI address within the specified space.

width is the write width.
    1:    Byte.
    2:    Word.
    4:    Longword.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

VXIoutReg

Writes a single word to a specified VXI register offset on the specified VXI device. The register is written in Motorola byte order and as nonprivileged data.
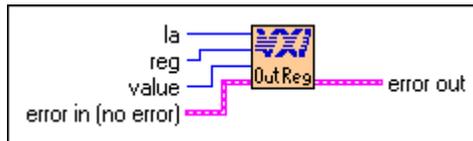
la is the logical address of the device to write to.

reg is the offset within VXI logical address registers.

value is the value written to the VXI register of the device.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

7

Local Resource Access VIs

This chapter describes the VIs you use to access miscellaneous local resources such as the local CPU VXI register set and Slot 0 MODID operations.

Access to the local logical address of the CPU is required for sending correct VXI signal values to other devices. Reading local VXI registers is required for retrieving configuration information. Exercising the local CPU MODID capabilities (if the local CPU is a VXI Slot 0 device) can be helpful in debugging the slot association (MODID) capability of a prototype VXI device.

Locating Local Resource Access VIs in LabVIEW

Select **Window»Show Diagram** to open the block diagram in LabVIEW. Select **Functions»Instrument I/O»I/O Compatibility»VXI»Local Resource Access** to locate the Local Resource Access VIs in LabVIEW.

Finding Help Online for Local Resource Access VIs

You can find helpful information about individual VIs online by using the LabVIEW Help window. Open the Help window by choosing **Help»Show Context Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You also can double-click the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

Handling Errors

The NI-VXI VIs use the LabVIEW error cluster method for handling errors. Each VI has error in and error out terminals. Refer to Appendix A, *Error Codes*, for information about how the error codes relate to the

previous status codes used by the NI-VXI VIs and a description of **error in** and **error out** parameters.

Local Resource Access VI Descriptions

This section describes the Local Resource Access VIs, which you can use to obtain the logical address of the local VXI device, sense and control the assertion of the MODID lines of the VXIbus backplane, and read or write a single byte or word from a particular VXI register.

GetMyLA

Gets the logical address of the local VXI device (the VXI device on which this copy of the NI-VXI software is running).

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

la is the logical address of the local device.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

ReadMODID

Senses the MODID lines of the VXIbus backplane. This VI applies only to the local device, which must be a Slot 0 device.
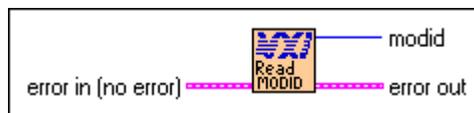
error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

| U16 | **modid** is a bit vector for Bits 12 through 0, corresponding to Slots 12 through 0, respectively. |

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetMODID

Controls the assertion of the MODID lines of the VXIbus backplane. This VI applies only to the local device, which must be a Slot 0 device.
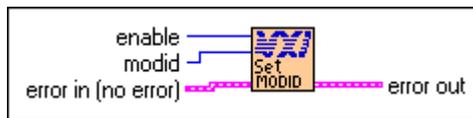
enable defines the handling of the MODID enable bit.
>   1:   Set MODID enable bit.
>   0:   Clear MODID enable bit.

modid is a bit vector for Bits 12 through 0, corresponding to Slots 12 through 0, respectively.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

VXIinLR

Reads a single byte, word, or longword from a particular VXI register on the local VXI device. The register is read in Motorola byte order and as nonprivileged data.

reg is the offset within VXI logical address registers.

| U16 |

width specifies byte, word, or longword.
- 1: Byte.
- 2: Word.
- 4: Longword.

| error in |

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

| U32 |

value is the data value read.

| error out |

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

VXIoutLR

Writes a single byte, word, or longword to a particular VXI register on the local VXI device. The register is written in Motorola byte order and as nonprivileged data.

| U16 |

reg is the offset within VXI logical address registers.

| U16 |

width specifies byte, word, or longword.
- 1: Byte.
- 2: Word.
- 4: Longword.

| U32 |

value is the data value to write.

| error in |

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

| error out |

**error out** contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

8

Shared Memory Access VIs

This chapter describes the VIs you use to perform shared memory operations. These VIs are useful for shared memory operation type communication. In shared memory applications, Local CPU RAM is shared on the VXI backplane.

On hardware platforms that support shared memory, you can use these VIs to lock down memory on your controller. Other VXI bus masters can then directly access this memory.

To set up shared memory, use the configuration utilities that shipped with your controller to request VXI address space from the Resource Manager. Some operating systems require you to lock down memory at system startup. You can do this by setting the Shared Memory Pool field in either of these utilities.

Locating Shared Memory Access VIs in LabVIEW

Select **Window»Show Diagrams** to open the block diagram in LabVIEW. Select **Functions»Instrument I/O»I/O Compatibility»VXI»Shared Memory Access** to locate the Local Resource Access VIs in LabView.

Finding Help Online for Shared Memory Access VIs

You can find helpful information about individual VIs online by using the LabVIEW Help window. Open the Help window by choosing **Help»Show Context Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window. You also can double-click the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

Handling Errors

The NI-VXI VIs now use the LabVIEW error cluster method for handling errors. Each VI has error in and error out terminals. Refer to Appendix A, *Error Codes*, for information about how the new error codes relate to the old status codes used by the NI-VXI VIs and a description of **error in** and **error out** parameters.

Shared Memory Resource VI Descriptions

This section describes the Shared Memory Resource VIs, which you can use to allocate, deallocate, and copy dynamic system RAM from the VXI Shared RAM area.

VXImemAlloc

Allocates dynamic system RAM from the VXI Shared RAM area of the local CPU and returns both the local and remote VXI addresses. The VXI address space is the same as the space for which the local device is porting memory. You can use this VI to set up shared memory transfers.
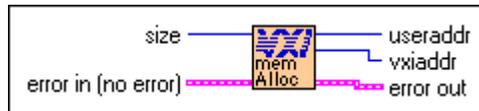
size is the number of bytes to allocate.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

useraddr is the returned application memory buffer address. This buffer cannot be directly accessed by LabVIEW. Use the VXImemCopy VI to access data in this buffer.

vxiaddr is the returned remote VXI memory buffer address.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

VXImemCopy

Copies data into the local memory array from the VXI Shared RAM area of the local CPU allocated by VXImemAlloc or copies data from the local memory array into the VXI Shared RAM.
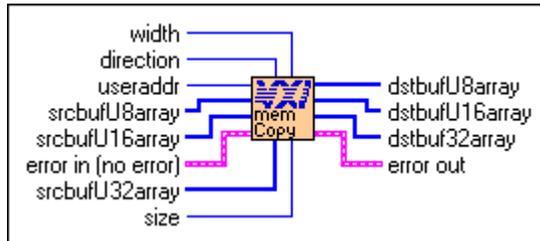
| U16 |

width is the copy width.
- 1:   Byte.
- 2:   Word.
- 4:   Longword.

| U16 |

direction designates the area from which you want to copy data.
- 1:   Copy data from the specified source array to the **useraddr** parameter.
- 0:   Copy data from the **useraddr** parameter into the destination array specified by the **width** parameter.

| U32 |

useraddr is the Shared RAM address to copy data from or into.

| [U8] |

srcbufU8array is the source unsigned byte array in the local address space (applicable only if the **direction** parameter indicates that you want to copy data from an array of bytes into the Shared RAM).

| [U16] |

srcbufU16array is the source unsigned word array in the local address space (applicable only if the **direction** parameter indicates that you want to copy data from an array of words into the Shared RAM).

| [P-B] |

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

| [U32] |

srcbufU32array is the source unsigned longword array in the local address space (applicable only if the **direction** parameter indicates that you want to copy data from an array of longwords into the Shared RAM).

| U32 |

size is the number of elements to copy.

dstbufU8array is the destination unsigned byte array in the local address space (applicable only if the **direction** parameter indicates that you want to copy data into an array of bytes from the Shared RAM and the **width** parameter indicates byte transfers).

dstbufU16array is the destination unsigned word array in the local address space (applicable only if the **direction** parameter indicates that you want to copy data into an array of words from the Shared RAM and the **width** parameter indicates word transfers).

dstbufU32array is the destination unsigned longword array in the local address space (applicable only if the **direction** parameter indicates that you want to copy data from an array of longwords from the Shared RAM and the **width** parameter indicates longword transfers).

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

VXImemFree

Deallocates dynamic system RAM from the VXI Shared RAM area of the local CPU that was allocated using the VXImemAlloc VI.
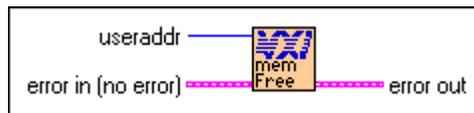
useraddr is the application memory buffer address to free.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

9

VXI Signal VIs

This chapter describes the VIs you use to specify signal routing, manipulate the global signal queue, and wait for a particular VXI signal to be received.

VXI signals are a basic form of asynchronous communication that VXI bus master devices use. A VXI signal is simply a 16-bit value written to the Signal register of a VXI Message-Based device. Normally, the write to the Signal register generates a local CPU interrupt, and the local CPU then acquires the signal value in some device-specific manner. All National Instruments hardware platforms have a hardware FIFO to accumulate signal values while waiting for the local CPU to retrieve them. The format of the 16-bit signal value is defined by the VXIbus specification. VXI signals and status/ID values contain the VXI logical address of the sending device in the lower 8 bits of the VXI signal or status/ID value. The upper 8 bits of the 16-bit value depends on the VXI device type.

Three methods are available for handling VXI signals in LabVIEW. One method is to allow the NI-VXI default signal handler to handle the signal. From the point of view of your LabVIEW application, using the default signal handler essentially results in the signal being ignored.

A second method for handling signals is by placing them into a global signal queue. The RouteSignal VI specifies which types of signals are handled by the default signal handler and which signals are placed on the global signal queue. (By default, when signals are initially enabled with the EnableSignalInt VI, all signals are routed to the default signal handler.) The VIs used to access the signal queue are SignalDeq, SignalEnq, and SignalJam.

The third method for handling signals is with the WaitForSignal VI. This VI can be used to suspend the execution of a VI until a particular signal (or one of a set of signals) arrives. In LabVIEW, any number of WaitForSignal VIs can be executed in parallel, even for the same logical address. When using the WaitForSignal VI, you should use RouteSignal to route the desired signals to the global signal queue. The WaitForSignal VI will first check the queue to see if the signal(s) in which you are interested have already been received.

Locating VXI Signal VIs in LabVIEW

Select **Window»Show Diagram** to open the block diagram in LabVIEW. Select **Functions»Instrument I/O»I/O Compatibility»VXI»VXI Signal** to locate the VXI Signal VIs in LabVIEW.

Finding Help Online for VXI Signal VIs

You can find helpful information about individual VIs online by using the LabVIEW Help window. Open the Help window by choosing **Help»Show Context Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You also can double-click the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

Handling Errors

The NI-VXI VIs use the LabVIEW error cluster method for handling errors. Each VI has error in and error out terminals. Refer to Appendix A, *Error Codes*, for information about how the error codes relate to the previous status codes used by the NI-VXI VIs and a description of **error in** and **error out** parameters.

VXI Signal VI Descriptions

This section describes VXI Signal VIs, which you can use to desensitize the local CPU to interrupts generated by writes to the local VXI Signal register, return the address of the current signal interrupt handler, specify how each tpe of signal is to be processed, and obtain a signal specified by the signal mask from the signal queue for the specified logical address.

DisableSignalInt

Desensitizes the local CPU to interrupts generated by writes to the local VXI Signal register. While disabled, no VXI signals are processed. If the local VXI hardware Signal register is implemented as a FIFO, signals are held in the FIFO until the signal interrupt is enabled via the EnableSignalInt VI.
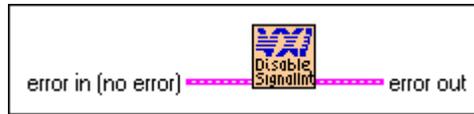
 error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

 error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

EnableSignalInt

Sensitizes the local CPU to interrupts generated by writes to the local VXI Signal register.

 error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

 error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetSignalHandler

Returns the address of the current signal interrupt handler for a specified logical address.
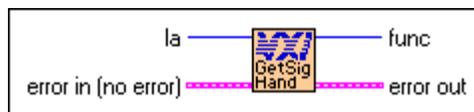
 la signifies the logical address for finding the address of the signal interrupt handler.

 – 2: Unknown (miscellaneous) signal source.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

func is the pointer to the current signal interrupt handler for the specified logical address.

    0:   Invalid **la**.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

RouteSignal

Specifies how each type of signal is to be processed for each logical address. A signal can be enqueued on a global signal queue (for later dequeuing via SignalDeq) or handled at interrupt service routine time by the default signal handler.
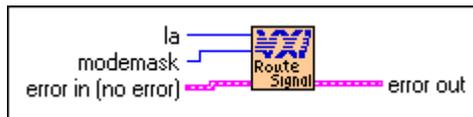
la specifies the routing for the logical address.

    –1: All known logical addresses.

modemask is 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 on the global signal queue, and all other signals are handled by the signal handler.

The following table describes the event signals that correspond to bits 14 through 8 when **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

Bit	Event Signal
10	Request False (REQF) Events
9	Request True (REQT) Events
8	No Cause Given Events

The following table shows the response signals that correspond to bits 7 through 0 when **la** is a Message-Based device.

Bit	Event Signal
7	Unused
6	B14 (reserved for future definition)
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 shows the type of signal values that correspond to bits 15 through 0 when **la** is a *non*-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)

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetSignalHandler

Replaces the current signal interrupt handler for a logical address with a specified handler.

I16
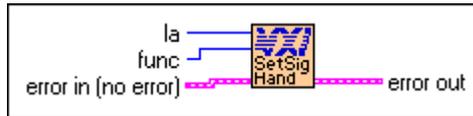
la specifies the logical address to set the handler to.
- –1: All known logical addresses.
- –2: Unknown (miscellaneous) signal handler.

U32

func is the pointer to the new signal interrupt handler.
- 0: Set to DefaultSignalHandler.
- 3: LabVIEW occurrence handler.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SignalDeq

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

I16
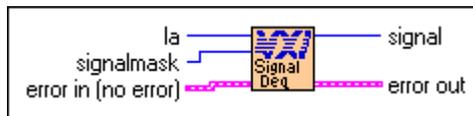
la specifies the logical address from which to dequeue signal.
- 255: VME interrupt routed to signal queue.
- –1: any known **la**.

U32

signalmask is a bit vector that specifies the types of signals to dequeue.

A 1 in any bit position causes the subroutine to dequeue signals of the associated type.

The following table describes the event signals that correspond to bits 14 through 8 when **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

The following table shows the response signals that correspond to bits 7 through 0 when **la** is a Message-Based device.

Bit	Event Signal
7	Unused
6	B14 (reserved for future definition)
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 shows the type of signal values that correspond to bits 15 through 0 when **la** is a *non*-Message-Based device, or if **la** = 255 (VME status/ID).

Bit	Event Signal
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)

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

signal is the signal value dequeued from the signal queue.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SignalEnq

Puts a signal on the tail of the signal queue for the specified logical address.
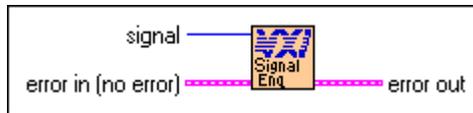
signal is the value to enqueue at the tail of the signal queue.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SignalHandler

Waits until a Signal interrupt occurs.
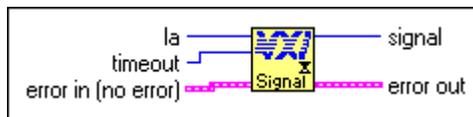
la specifies the logical address of the signals to be handled
    –1: All known logical addresses.
    –2: Unknown (miscellaneous) signal sources.

timeout specifies the number of milliseconds to wait for the interrupt
 –1: Forever.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

signal specifies the actual 16-bit VXI signal received.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SignalJam

Puts a signal on the head of the signal queue for the specified logical address.
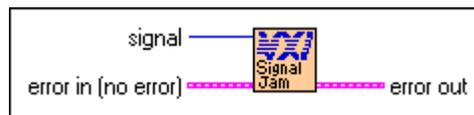
Note This VI is intended for debugging purposes only.

**signal** is the signal value to put on the head of the queue.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WaitForSignal

Waits for a specified type of signal to be received from a specified logical address. A signal mask defines the type(s) of signals that the application program waits for. The timeout value specifies the maximum amount of time (in milliseconds) to wait until the signal occurs.
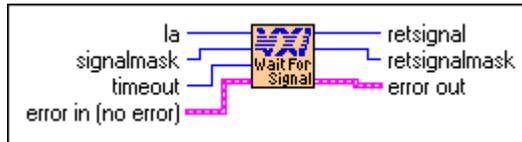
| `I16` | **la** is the logical address of the device sourcing the signal.
255: VME interrupt routed to signal queue.
−1: any known **la**. |

| `U32` | **signalmask** is a bit vector that indicates the type(s) of signals that the application will wait for. |

A one in any bit position causes the subroutine to detect signals of the associated type.

The following table shows the event signals that correspond to bits 14 through 8, when **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

The following table shows the response signals that correspond to bits 7 through 0, when **la** is a Message-Based device.

Bit	Event Signal
7	Unused
6	B14 (reserved for future definition)
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 shows the type of signal values that correspond to bits 15 through 0 when **la** is a *non*-Message-Based device, or if **la** = 255 (VME status/ID).

Bit	Event Signal
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)

I32

timeout is the maximum amount of time (in milliseconds) to wait until the signal occurs.
    0:   Forever.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

U16

retsignal is the signal received (upper byte of the 16-bit signal).

U32

retsignalmask is a bit vector that indicates the type(s) of signals that the application received. The bits have the same meanings as given for the input parameter **signalmask**.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

10

VXI Interrupt VIs

This chapter describes the VIs that control VXI interrupts. VXI interrupts are a basic form of asynchronous communication used by VXI devices with VXI interrupter support. In VME, a device asserts a VME interrupt line, and the VME interrupt handler device acknowledges the interrupt. During the VME interrupt acknowledge cycle, a status/ID value is returned.

On most 680x0-based VME CPUs, this value is used as a local interrupt vector value and routed directly to the 680x0 processor. This value is used to look up which interrupt service routine to invoke. In VXI, however, the VXI interrupt acknowledge cycle returns (at a minimum) a 16-bit status/ID value. This 16-bit status/ID value is data, not a vector base location. The definition of the 16-bit vector is specified by the VXIbus specification and is the same as for a VXI signal. The lower 8 bits of the status/ID value is the VXI logical address of the interrupting device, while the upper 8 bits specifies the reason for interrupting.

Because the interrupt status/ID value for a VXI device is the same as a VXI signal value, VXI interrupts can be handled as VXI signals. The RouteVXIint VI is used to specify that a VXI interrupt be handled as a VXI signal. If a VXI interrupt is not routed to be processed as a VXI signal, the interrupt will be handled by the NI-VXI default VXI interrupt handler, resulting in your LabVIEW application essentially ignoring the interrupt.

To sensitize and desensitize the LabVIEW application to VXI interrupts routed to the VXI interrupt handlers, use the EnableVXIint and DisableVXIint VIs, respectively. To enable queuing of the interrupt after executing RouteVXIint, you must execute the EnableVXItoSignalInt VI. Once the VXI interrupt is routed and enabled, the interrupt is handled as a VXI signal using the signal queue VIs. To disable queuing of the interrupt, you must execute the DisableVXItoSignalInt VI. For more information about VXI signal VIs, refer to Chapter 9, *VXI Signal VIs*.

Locating VXI Interrupt VIs in LabVIEW

Select **Window»Show Diagram** to open the block diagram in LabVIEW. Select **Functions»Instrument I/O»I/O Compatibility»VXI»VXI Interrupt** to locate the VXI Interrupt VIs in LabVIEW.

Finding Help Online for VXI Interrupt VIs

You can find helpful information about individual VIs online by using the LabVIEW Help window. Open the Help window by choosing **Help»Show Context Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You also can double-click the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

Handling Errors

The NI-VXI VIs use the LabVIEW error cluster method for handling errors. Each VI has error in and error out terminals. Refer to Appendix A, *Error Codes*, for information about how the error codes relate to the previous status codes used by the NI-VXI VIs and a description of **error in** and **error out** parameters.

VXI Interrupt VI Descriptions

This section describes the VXI Interrupt VIs, which you can use to perform an IACK cycle on the VXIbus on the specified controller, assert and asynchronously deassert a VXI interrupt line, sensitize and desensitize the local CPU, and return the address of the current interrupt handler.

AcknowledgeVXIint

Performs an IACK cycle on the VXIbus on the specified controller (either an embedded CPU or an extended controller) for a particular VXI interrupt level.
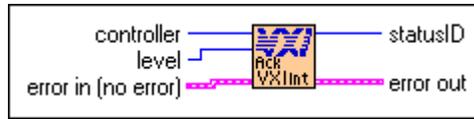
VXI interrupts are automatically acknowledged when enabled by the EnableVXItoSignalInt and EnableVXIint VIs. Use this VI to manually acknowledge VXI interrupts that the local device is not enabled to receive.

✏️   **Note**   This VI is intended for debugging purposes only.

I16	**controller** is the controller on which to acknowledge the interrupt.
U16	**level** is the interrupt level to acknowledge.
[error]	**error in** describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.
U32	**statusID** is the status/ID obtained during the IACK cycle.
[error]	**error out** contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

AssertVXIint

Asserts a VXI interrupt line on the specified controller (either an embedded CPU or an extended controller).
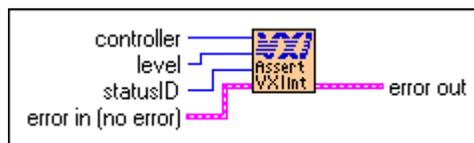
When the VXI interrupt is acknowledged (a VXI IACK cycle occurs), the specified status/ID is passed to the device that acknowledges the VXI interrupt.

I16	**controller** is the controller on which to assert the interrupt.
U16	**level** is the interrupt level to assert.
U32	**statusID** is the status/ID to present during the IACK cycle.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

DeAssertVXIint

Asynchronously deasserts a VXI interrupt line on the specified controller (either an embedded CPU or an extended controller) that was previously asserted by the AssertVXIint VI.
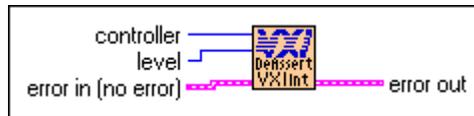
Note This VI is intended for debugging purposes only. Deasserting a VXI interrupt can cause a violation of the VME and VXIbus specifications.

controller is the controller on which to deassert the interrupt.

level is the interrupt level to deassert.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

DisableVXIint

Desensitizes the local CPU to specified VXI interrupts generated in the specified controller, which are routed to be handled as VXI interrupts (not as signals) via the RouteVXIint VI.

controller specifies the controller (embedded or extended) to disable the interrupts.

levels is a vector of VXI interrupt levels to disable.

Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.
- 1: Disable for appropriate level.
- 0: Leave at current setting.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

DisableVXItoSignalInt

Desensitizes the local CPU to specified VXI interrupts generated in the specified controller, which are routed to be handled as VXI signals (not as interrupts) via the RouteVXIint VI.

controller specifies the controller (embedded or extended) to disable the interrupts.

levels is a vector of VXI interrupt levels to disable.
Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.
- 1: Disable for appropriate level.
- 0: Leave at current setting.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

EnableVXIint

Sensitizes the local CPU to specified VXI interrupts generated in the specified controller, which are routed to be handled as VXI interrupts (not as signals) via the RouteVXIint VI.
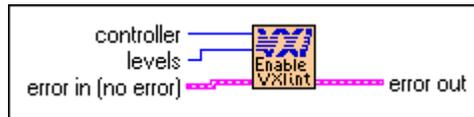
The RM assigns the interrupt levels automatically. Use the GetDevInfoShort VI to retrieve the assigned levels. Notice that each VXI interrupt is physically enabled only if the RouteVXIint VI has specified that the VXI interrupt be routed and then handled as a VXI/VME interrupt.

controller specifies the controller (embedded or extended) to enable the interrupts.

levels is a vector of VXI interrupt levels to enable.
Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.
- 1: Enable for appropriate level.
- 0: Leave at current setting.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

EnableVXItoSignalInt

Sensitizes the local CPU to specified VXI interrupts generated in the specified controller, which are routed to be handled as VXI signals (not as interrupts) via the RouteVXIint VI.
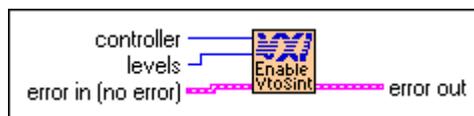
The RM assigns the interrupt levels automatically. GetDevInfoShort can be used to retrieve the assigned levels. Notice that each VXI interrupt is physically enabled only if the RouteVXIint VI has specified that the VXI interrupt be routed and then handled as a VXI/VME signal.

| I16 | **controller** specifies the controller (embedded or extended) to enable the interrupts. |

| U16 | **levels** is a vector of VXI interrupt levels to enable.
Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.
1:   Enable for appropriate level.
0:   Leave at current setting. |

| [error] | **error in** describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors. |

| [error] | **error out** contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information. |

GetVXIintHandler

Returns the address of the current interrupt handler for a specified VXIbus interrupt level.
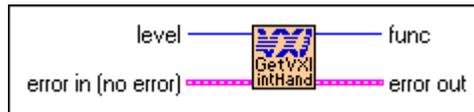
| U16 | **level** is the VXI interrupt level associated with the handler. |

| [error] | **error in** describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors. |

| U32 | **func** is the pointer to the current interrupt handler for a specified VXIbus interrupt level.
0:   Invalid **level** or no hardware support. |

| [error] | **error out** contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information. |

RouteVXIint

Specifies whether the status/ID value retrieved from a VXI interrupt acknowledge cycle is routed to the VXI interrupt handler or to the signal processing routine.
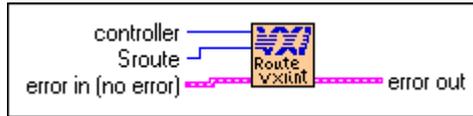
The RouteVXIint VI dynamically enables and disables the appropriate VXI interrupts based on the settings from calls to the EnableVXItoSignalInt and EnableVXIint VIs.

controller specifies the controller (embedded or extended) to enable the interrupts.

Sroute is a bit vector that specifies whether to handle a VXI interrupt as a signal or route it to the VXI interrupt handler routine.

>   Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.
>
>>   1: Handle VXI interrupt for this level as a signal.
>>   0: Handle VXI interrupt as a VXI interrupt.
>
>   Bits 14 to 8 correspond to VXI interrupt levels 7 to 1, respectively.
>
>>   1: Route as 8-bit VME status/ID.
>>   0: Route as 16-bit VXI status/ID.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetVXIintHandler

Replaces the current interrupt handler for the specified VXIbus interrupt levels with a specified handler.
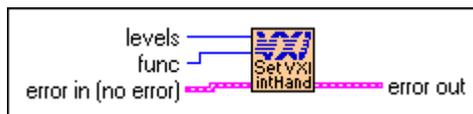
U16	**levels** is a bit vector of VXI interrupt levels. Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.

        1:    Set.
        0:    Do not set handler.

U32	**func** is the pointer to the new VXI interrupt handler.

        0:    Set to DefaultVXIintHandler.
   3:   LabVIEW occurrence handler.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

VXIintAcknowledgeMode

Specifies whether to handle the VXI interrupt acknowledge cycle for the specified controller (embedded or extended) for the specified levels as Release on Acknowledge (ROAK) interrupts or as Release on Register Access (RORA) interrupts.
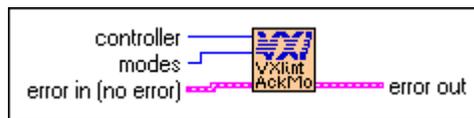
If the VXI interrupt level is handled as a RORA VXI interrupt, the local interrupt generation is automatically inhibited when the VXI interrupt acknowledge is performed. The EnableVXIint or EnableVXItoSignalInt VIs must be called to reenable the appropriate VXI interrupt level whenever a RORA VXI interrupt occurs.

I16	**controller** specifies the controller (embedded or extended) for which to specify the routing.

U16	**modes** is a vector of VXI interrupt levels to set to RORA/ROAK interrupt mode.

Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.
        1:    Set to RORA VXI interrupt for corresponding level.
        0:    Set to ROAK VXI interrupt for corresponding level.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

VXIintHandler

Waits until a VXI interrupt occurs on the specified level(s).
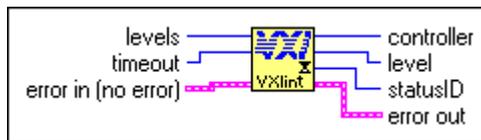
levels is a bit vector that specifies which level to poll.
  - 1: Set.
  - 0: Do not set.

Bits O to 6 correspond to VXI interrupt levels 1 to 7, respectively.

timeout specifies the number of milliseconds to wait for the interrupt.
  - –1 Forever.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

controller specifies the logical address of controller interrupting.

level specifies the received VXI interrupt level.

statusID specifies the status/ID obtained during IACK cycle. (If **statusID** is a 16-bit VXI IACK value, it may be equivalent to a VXI signal.)

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

11

VXI Trigger VIs

This chapter describes the VIs that control triggers, a backplane feature that VXI adds to the VME standard. You can use four basic protocols (SYNC, ASYNC, SEMI-SYNC, and START/STOP) for device synchronization, for stepping through tests, or for a command path. The Trigger VIs fall into four categories:

- Source Trigger VIs act as a standard interface for asserting (sourcing) TTL and ECL triggers, as well as for detecting acknowledgements from accepting devices.

- Acceptor Trigger VIs act as a standard interface for sensing (accepting) TTL and ECL triggers, as well as for sending acknowledgements back to the sourcing device.

- Map Trigger VIs act as configuration tools for multiframe and local support for VXI triggers.

- Trigger Configuration VIs configure not only the general settings of the trigger inputs and outputs, but also the TIC counter and tick timers.

The actual capabilities of specific systems are based on the triggering capabilities of the hardware devices involved (both the sourcing and accepting devices).

Locating VXI Trigger VIs in LabVIEW

Select **Window»Show Diagram** to open the block diagram in LabVIEW. Select **Functions»Instrument I/O»I/O Compatibility»VXI»VXI Trigger** to locate the VXI Trigger VIs in LabVIEW.

Finding Help Online for VXI Trigger VIs

You can find helpful information about individual VIs online by using the LabVIEW Help window. Open the Help window by choosing **Help»Show Context Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You also can double-click the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

Handling Errors

The NI-VXI VIs use the LabVIEW error cluster method for handling errors. Each VI has error in and error out terminals. Refer to Appendix A, *Error Codes*, for information about how the error codes relate to the previous status codes used by the NI-VXI VIs and a description of **error in** and **error out** parameters.

VXI Trigger VI Descriptions

This section describes VXI Trigger VIs, which you can use to acknowledge the specified TTL/ECL or external (GPIO) trigger, disable the sensing of the specified TTL/ECL trigger line, counter, or tick timer, enable the sensing of the specified TTL/ECL trigger line, return the address of the current TTL/ECL trigger, and map specified TTL, ECL, Star X, Stary Y or miscellaneous signal line to another.

AcknowledgeTrig

Acknowledges the specified TTL/ECL or external (GPIO) trigger on the specified controller.
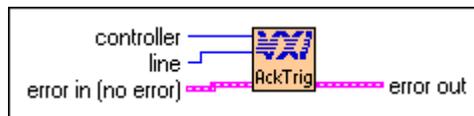
The TTL/ECL trigger interrupt handler is called after an TTL/ECL trigger is sensed. If the sensed protocol requires an acknowledge (ASYNC or SEMI-SYNC protocols), the application should call the AcknowledgeTrig VI after performing any device-dependent operations. If you configured a trigger line using the TrigAssertConfig VI to participate in

external (GPIO) SEMI-SYNC acknowledging, you can use the AcknowledgeTrig VI to manually acknowledge a pending external SEMI-SYNC trigger.

| I16 | **controller** is the controller on which to acknowledge the trigger interrupt. |

| U16 | **line** is the TTL, ECL, or external trigger line to acknowledge. Refer to the following table for the meaning of the values. |

Value	Trigger Lines
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)

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

DisableTrigSense

Disables the sensing of the specified TTL/ECL trigger line, counter, or tick timer that was enabled by the EnableTrigSense VI.
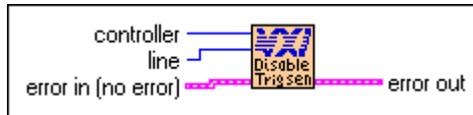
controller is the controller on which to disable sensing.

line is the trigger line to disable sensing.

Value	Trigger Lines
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*
* Only with controllers that have the TIC ASIC (Application Specific Integrated Circuit).	

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

EnableTrigSense

Enables the sensing of the specified TTL/ECL trigger line or starts up the counter or tick timer for the specified protocol.
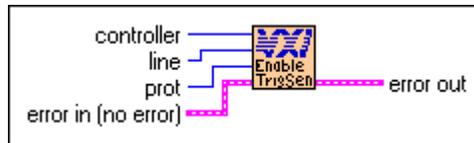
When the protocol is sensed, the corresponding trigger interrupt handler will be invoked. In order to start up the counter or tick timers, you must first call either the TrigCntrConfig or TrigTickConfig VIs, respectively.

controller is the controller on which to enable sensing.

line is the trigger line to enable sensing. Refer to the following table for the meaning of the values.

Value	Trigger Lines
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*
* Only with controllers that have the TIC ASIC.	

U16

prot specifies the protocol to use.
- 2: START
- 3: STOP
- 4: SYNC
- 5: SEMI-SYNC
- 6: ASYNC

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetTrigHandler

Returns the address of the current TTL/ECL trigger, counter, or tick timer interrupt handler for a specified trigger source.

U16

line is the TTL, ECL trigger line or counter/tick. Refer to the following table for the meaning of the values.

Value	Trigger Lines
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*
* Only with controllers that have the TIC ASIC.	

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

func is a pointer to the current trigger interrupt handler for a specified trigger line to be used with the SetTrigHandler VI.

    0:    Invalid **line** or no hardware support.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

MapTrigToTrig

Maps the specified TTL, ECL, Star X, Star Y, external connection (GPIO), or miscellaneous signal line to another.
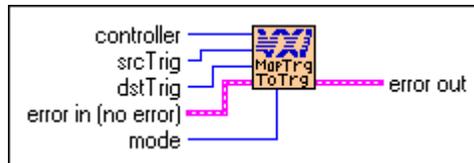
The support actually present is completely hardware dependent and is reflected in the error status and in hardware-specific documentation.

controller is the controller on which to map signal lines.

srcTrig is the source line to map to destination line.

dstTrig is the destination line to map from source line. Refer to the following table for the meaning of the values. (Star X and Star Y are not currently supported lines.)

| Value	Source or Destination
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
14 to 26	Star X lines 0 to 12
27 to 39	Star Y lines 0 to 12
40 to 49	External source/destination (GPIO 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 GPIO 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*
* Only with controllers that have the TIC ASIC.	

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

mode is the signal conditioning mode.
    0:   No conditioning.

Bits 0 through 3 have the following conditioning effects.

Bit	Conditioning Effect
0	Synchronize with next CLK edge.*
1	Invert signal polarity
2	Pulse stretch to one CLK minimum.*
3	Use EXTCLK (not CLK10) for conditioning
* Only with controllers that have the TIC ASIC.	

All other values are reserved for future expansion.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetTrigHandler

Replaces the current TTL/ECL trigger, counter, or tick timer interrupt handler for a specified trigger source with the specified function, **func**.
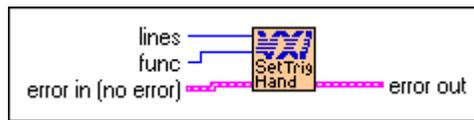
lines is a bit vector of trigger lines.
   1:   Set.
   0:   Do not set.

Bit	Trigger Lines
0 to 7	TTL lines 0 to 7
8 to 13	ECL lines 0 to 5
14	TIC counter*
15	TIC TICK timers*
* Only with controllers that have the TIC ASIC.	

func is a pointer to the new trigger interrupt handler returned from the GetTrigHandler VI.
   0:   DefaultTrigHandler. (Calls AcknowledgeTrig if the interrupt is received while enabled to sense a trigger line using EnableTrigSense.)
   1:   DefaultTrigHandler2. (Does not call AcknowledgeTrig. The user is responsible for calling AcknowledgeTrig.)
   3:   LabVIEW Occurrence Handler.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.


error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SrcTrig

Sources the specified protocol on the specified TTL, ECL, or external trigger line on the specified controller.
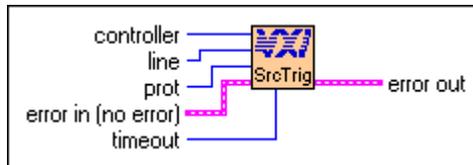
controller is the controller on which to source the trigger line.

line is the trigger line to source. Refer to the following table for the meaning of the values.

Value	Trigger Lines
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
* Only with controllers that have the TIC ASIC.	

prot specifies the protocol to use.
    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).

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

timeout specifies the timeout value in milliseconds.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

TrigAssertConfig

Configures the specified TTL/ECL trigger line assertion method. TTL/ECL triggers can be (re-) synchronized to CLK10 on a per line basis. You can globally select all TTL/ECL trigger lines to synchronize to either the rising or falling edge of CLK10. In addition, you can specify a trigger line specified to partake in SEMI-SYNC accepting with external acknowledge.
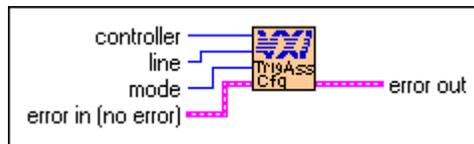
controller is the controller on which to configure assertion mode.

line is the trigger line to configure. Refer to the following table for the meaning of the values.

Value	Trigger Lines
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
FFFFH	General assertion configuration (all lines).

mode specifies the configuration mode.

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

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

All other values are reserved for future expansion.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

TrigCntrConfig

Configures TIC chip internal 16-bit counter. Call SrcTrig or EnableTrigSense to actually start the counter. The input can be any trigger line, CLK10, or the EXTCLK connection. This VI works only with controllers that have the TIC ASIC.
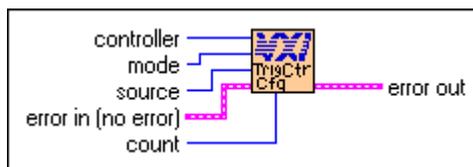
The counter has two outputs: TCNTR (one 100-nsec pulse per input edge) and GCNTR (unasserted until count goes from 1 to 0, then asserted until counter reloaded or reset). Use the MapTrigToTrig VI to map TCNTR to any number of the TTL or ECL trigger lines and to map GCNTR to any number of the external (GPIO) lines.

controller is the controller on which to configure the TIC counter.

U16

mode is the configuration mode.
- 0:   Initialize the counter.
- 2:   Reload the counter leaving enabled.
- 3:   Disable/abort any count in progress

U16

source is the trigger line to configure as input to counter.

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

U16

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

count specifies the number of input pulses to count before terminating.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

TrigExtConfig

Configures the external trigger (GPIO) lines. You can feed back the external trigger lines for use in the crosspoint switch output. You can assert the external trigger lines high or low, or leave them unconfigured (tristated) for use as a crosspoint switch input. If the external trigger lines are not feedback, you can invert the external input before mapping it to a trigger line.

I16
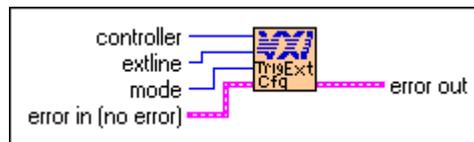
controller is the controller on which to configure the external connection.

U16

extline is the trigger line to configure. Refer to the following table for the meaning of the values.

Value	Trigger Lines
40 to 49	External source/destination (GPIO 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 GPIO 4 to 9

U16

mode specifies the configuration mode.

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

All other values are reserved for future expansion.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

TrigHandler

Waits until a trigger interrupt occurs on the specified line(s).
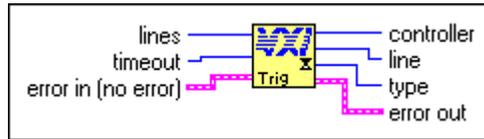
`[U16]`

lines is a bit vector that specifies which lines to poll.
- 1:   Set.
- 0:   Do not set.

Bit	Trigger Lines
0 to 7	TTL lines 0 to 7
8 to 13	ECL lines 0 to 5
14	TIC counter
15	TIC TICK timers

`[I32]`

timeout specifies the number of milliseconds to wait for the interrupt.
- −1:   Forever.

`[error]`

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

`[I16]`

controller is the controller form which the trigger interrupt is received.

`[U16]`

line is the trigger line on which the trigger interrupt was received. Refer to the following table for the meaning of the values.

Value	Trigger Lines
0 to 7	TTL lines 0 to 7
8 to 13	ECL lines 0 to 5
50	TIC counter
60	TIC TICK timers

`[U16]`

type is a bit vector that specifies the type of trigger interrupt.

Bit	Trigger Lines
0	1: Source trigger acknowledged. 0: Trigger sensed.
2	1: Assertion edge overrun occurred.
3	1: Unassertion edge overrun occurred.
4	1: Pulse stretch overrun occurred.
5	1: Error summary (2, 3, 4:1).

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

TrigTickConfig

Configures TIC chip internal dual 5-bit tick timers. Call the SrcTrig or EnableTrigSense VIs to actually start the tick timers. The SrcTrig VI inhibits the TICK1 output from generating tick timer interrupts. The EnableTrigSense VI enables the TICK1 output to generate tick timer interrupts. The input can be any external (GPIO) line, CLK10, or the EXTCLK connection. You can map the two tick timer outputs, TICK1 and TICK2, to any number of TTL/ECL trigger lines. In addition, you can map the TICK2 output to any number of external (GPIO) lines. This VI works only with controllers that have the TIC ASIC.
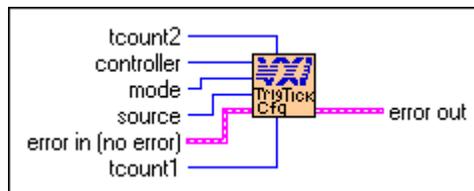
tcount2 is the number of input pulses (as a power of two) to count before asserting TICK2 output.

controller is the controller on which to configure the TIC chip dual 5-bit tick timers.

mode is the 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.

U16

source is the trigger line to configure as input to counter. Refer to the following table for the meaning of the values.

| Value	Configuration Modes
40 to 49	External source/destination (GPIO 0 to 9)
40	Front panel in (connector 1)
41	Front panel out (connector 2)
70	CLK10
71	EXTCLK connection

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

U16

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

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

UnMapTrigToTrig

Unmaps the specified TTL, ECL, Star X, Star Y, external connection (GPIO), or miscellaneous signal line that was mapped to another line using the MapTrigToTrig VI.

I16
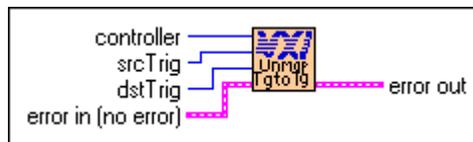
controller is the controller on which to unmap signal lines.

U16

srcTrig is the source line to unmap from destination line.

U16

dstTrig is the destination line mapped from source line. Refer to the following table for the meaning of the values.

Value	Source or Destination
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
14 to 26	Star X lines 0 to 12
27 to 39	Star Y lines 0 to 12
40 to 49	External source/destination (GPIO 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 GPIOs 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*
* Only with controllers that have the TIC ASIC.	

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

WaitForTrig

Waits for the specified trigger line to be sensed on the specified controller for the specified time. The EnableTrigSense VI must be called to sensitize the hardware to the particular trigger protocol to be sensed.
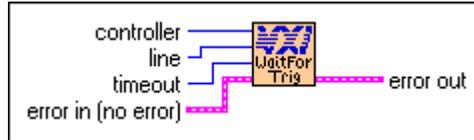
controller is the controller on which to wait for trigger.

line is the trigger line to wait on. Refer to the following table for the meaning of the values.

Value	Trigger Lines
0 to 7	TTL trigger lines 0 to 7
8 to 13	ECL trigger lines 0 to 5
50	TIC counter*
60	TIC TICK1 tick timer*
* Only with controllers that have the TIC ASIC.	

timeout specifies the timeout value (in milliseconds).

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

**error out** contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

12

System Interrupt Handler VIs

This chapter describes the System Interrupt Handler VIs. You can use these VIs to handle miscellaneous system conditions that can occur in the VXI environment, such as Sysfail, ACfail, BusError, Sysreset, or Soft Reset conditions. The NI-VXI software interface can handle these system conditions for the application through the use of default interrupt service routines.

The NI-VXI software handles all system interrupt handlers in the same manner. Each type of interrupt has its own specified default handler, which is installed when InitVXIlibrary initializes the NI-VXI software. All system interrupt handlers are initially disabled (except for BusError). The corresponding enable function for each handler must be called in order to invoke the default handler.

Locating System Interrupt Handler VIs in LabVIEW

Select **Window»Show Diagram** to open the block diagram in LabVIEW. Select **Functions»Instrument I/O»I/O Compatibility»VXI»System Interrupt Handler** to locate the System Interrupt Handler VIs in LabVIEW.

Finding Help Online for System Interrupt Handler VIs

You can find helpful information about individual VIs online by using the LabVIEW Help window. Open the Help window by choosing **Help»Show Context Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You also can double-click the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

Handling Errors

The NI-VXI VIs use the LabVIEW error cluster method for handling errors. Each VI has error in and error out terminals. Refer to Appendix A, *Error Codes*, for information about how the error codes relate to the previous status codes used by the NI-VXI VIs and a description of **error in** and **error out** parameters.

System Interrupt Handler VI Descriptions

This section describes System Interrupt Handler VIs, which you can use to assert the SYSRESET* signal in the mainframe specified by **controller**, desensitize the local CPU from interrupts generated from ACfail conditions, enable or disable the local Soft Reset interrupt, and return the address of an interrupt handler.

ACfailHandler

Waits until an ACfail interrupt occurs.

ACfailHandler.bmp
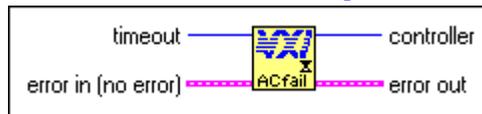
timeout specifies the number of milliseconds to wait for the interrupt.
   –1: Forever.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

controller specifies the logical address of controller interrupting.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

AssertSysreset

Asserts the SYSRESET* signal in the mainframe specified by **controller**.

AssertSysreset.bmp

I16
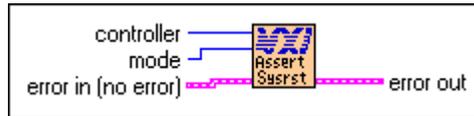
controller specifies the logical address of the mainframe extender on which to assert SYSRESET*.
- –1: From the local CPU or first extended controller.
- –2: All extenders.

U16

mode specifies the mode of execution.
- 0: Do not disturb original configuration.
- 1: Force link between SYSRESET* and local reset (SYSRESET* resets local CPU).
- 2: Break link between SYSRESET* and local reset (SYSRESET* does *not* reset local CPU).

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

BusErrorHandler

Waits until a bus error occurs.

BusErrorHandler.bmp
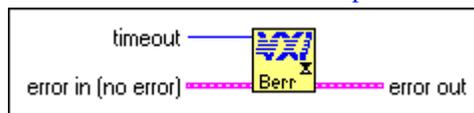
I32

timeout specifies the number of milliseconds to wait for the interrupt.
- –1: Forever.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

DisableACfail

Desensitizes the local CPU from interrupts generated from ACfail conditions on the embedded CPU VXIbus backplane, or from the specified extended controller VXI backplane (if external CPU).

DisableAcfail.bmp
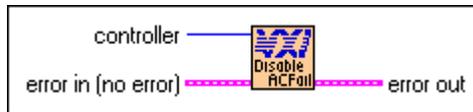
controller specifies the logical address of the mainframe extender to disable.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

DisableSoftReset

Disables the local Soft Reset interrupt being generated from a write to the reset bit of the local CPU control register.

DisableSoftReset.bmp
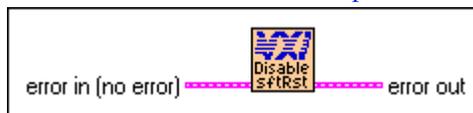
error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

DisableSysfail

Desensitizes the local CPU from interrupts generated from Sysfail conditions on the embedded CPU VXIbus backplane or from the specified extended controller VXI backplane (if external CPU).

DisableSysfail.bmp

controller specifies the logical address of the mainframe extender to disable.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

DisableSysreset

Desensitizes the application from Sysreset interrupts generated from the embedded CPU VXIbus backplane or from the specified extended controller VXI backplane (if external CPU).

DisableSysreset.bmp

controller specifies the logical address of the mainframe extender to disable.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

EnableACfail

Sensitizes the local CPU to interrupts generated from ACfail conditions on the embedded CPU's VXIbus backplane or from the specified controller's VXI backplane (if external CPU).

EnableACfail.bmp
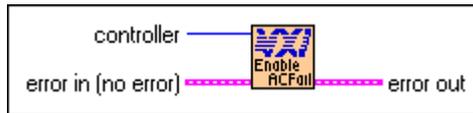
controller specifies the logical address of the mainframe extender to enable.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.


error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

EnableSoftReset

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

EnableSoftreset.bmp
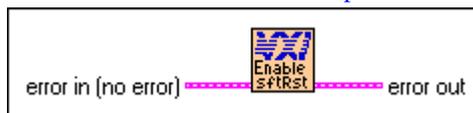
error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

EnableSysfail

Sensitizes the local CPU to interrupts generated from Sysfail conditions on the embedded CPU's VXIbus backplane or from the specified extended controller's VXI backplane (if external CPU).

EnableSysfail.bmp

controller specifies the logical address of the mainframe extender to enable.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

EnableSysreset

Sensitizes the local CPU to Sysreset interrupts generated from the embedded CPU's VXIbus backplane or from the specified extended controller's VXI backplane (if external CPU).

EnableSysreset.bmp

controller specifies the logical address of the mainframe extender to enable.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetACfailHandler

Returns the address of the current ACfail interrupt handler.

GetACfailHandler.bmp

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

func is the pointer to the current ACfail interrupt handler.
    0:   ACfail interrupts not supported.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetBusErrorHandler

Returns the address of the current user Bus Error interrupt handler.

GetBusErrorHandler.bmp

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

func is the pointer to the current Bus Error interrupt handler.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetSoftResetHandler

Returns the address of the current Soft Reset interrupt handler.

GetSoftResetHandler.bmp

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

func is the pointer to the current Soft Reset interrupt handler.
    0:   Soft Reset interrupts not supported.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetSysfailHandler

Returns the address of the current Sysfail interrupt handler.

GetSysfailHandler.bmp

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

func is the pointer to the current Sysfail interrupt handler.
    0:   Sysfail interrupts not supported.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

GetSysresetHandler

Returns the address of the current SYSRESET* interrupt handler.

GetSysResetHandler.bmp

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

func is the pointer to the current SYSRESET* interrupt handler.
    0:    SYSRESET* interrupts not supported.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetACfailHandler

Replaces the current ACfail interrupt handler with a specified handler.

SetACFa.bmp
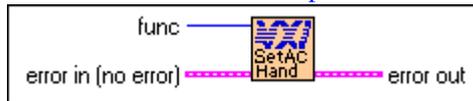
func points to the new ACfail interrupt handler.
    0:    Set to DefaultACfailHandler.
    3:    LabVIEW occurrence handler.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetBusErrorHandler

Replaces the current Bus Error handler with a specified handler.

SetBusErrorHandler.bmp

func points to the new Bus Error interrupt handler.
- 0:   Set to DefaultBusErrorHandler.
- 3:   LabVIEW occurrence handler.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetSoftResetHandler

Replaces the current Soft Reset interrupt handler with a specified handler.

SetSoftResetHandler.bmp
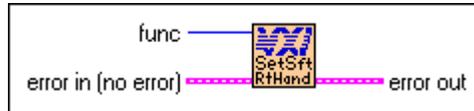
func points to the new Soft Reset interrupt handler.
- 0:   Set to DefaultSoftResetHandler.
- 3:   LabVIEW occurrence handler.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetSysfailHandler

Replaces the current Sysfail interrupt handler with a specified function, **func**.

SetSysfailHandler.bmp
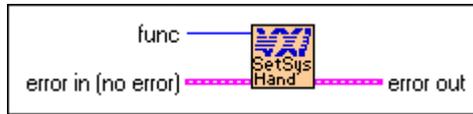
func is a pointer to the new Sysfail handler.

- 0:   DefaultSysfailHandler sets only the Sysfail Inhibit bit in the Control register of the failed Servant.
- 1:   DefaultSysfailHandler2 sets the Reset bit along with the Sysfail Inhibit bit in the Control register of the failed Servant.
- 3:   LabVIEW occurrence handler.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SetSysresetHandler

Replaces the current SYSRESET* interrupt handler with a specified handler, **func**.

SetSysresetHandler.bmp
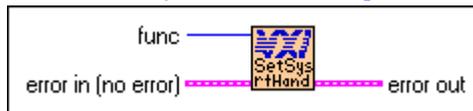
func is the pointer to the new SYSRESET* interrupt handler.

- 0:   Set to DefaultSysresetHandler.
- 3:   LabVIEW occurrence handler.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SoftResetHandler

Waits until a SoftReset interrupt occurs.

SoftResetHandler.bmp

I32

timeout specifies the number of milliseconds to wait for the interrupt.
   –1: Forever.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SysfailHandler

Waits until a Sysfail interrupt occurs.

SysfailHandler.bmp
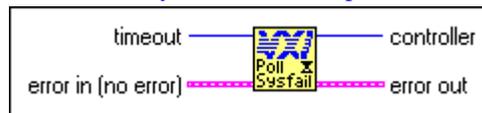
I32

timeout specifies the number of milliseconds to wait for the interrupt.
   –1: Forever.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

I16

controller specifies the logical address of controller interrupting.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

SysresetHandler

Waits until a SYSRESET* interrupt occurs.

SysresetHandler.bmp

timeout specifies the number of milliseconds to wait for the interrupt.
  –1: Forever.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

controller specifies the logical address of controller interrupting.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

13

VXIbus Extender VIs

This chapter describes the VXIbus Extender VIs. The NI-VXI software interface fully supports the standard VXIbus extension method presented in the *VXIbus Mainframe Extender Specification*. When the National Instruments Resource Manager completes its configuration, all default transparent extensions are complete. You can use these VIs to dynamically change these extensions if your application has such a requirement.

The transparent extensions include extensions of VXI Interrupt, TTL trigger, ECL trigger, Sysfail, ACfail, and Sysreset VXIbus signals for multi-mainframe systems. You can use these VIs to dynamically change these extensions if your application has such requirements. Usually, the application will never need to change the default settings. Consult your utilities manual on how to use vxiedit, the NI-VXI resource program editor, to change the default extender settings.

Locating VXIbus Extender VIs in LabVIEW

Select **Window»Show Diagram** to open the block diagram in LabVIEW. Select **Functions»Instrument I/O»I/O Compatibility»VXI»VXIbus Extender** to locate the VXIbus Extender VIs in LabVIEW.

Finding Help Online for VXIbus Extender VIs

You can find helpful information about individual VIs online by using the LabVIEW Help window. Open the Help window by choosing **Help»Show Context Help** in LabVIEW. When you place the cursor on a VI icon, the wiring diagram and parameter names for that VI appear in the Help window.

You also can double-click the VI to open the front panel. When the Help window is open, you can get more information on each parameter by placing the cursor over the corresponding control or indicator on the VI front panel.

Handling Errors

The NI-VXI VIs use the LabVIEW error cluster method for handling errors. Each VI has error in and error out terminals. Refer to Appendix A, *Error Codes*, for information about how the error codes relate to the previous status codes used by the NI-VXI VIs and a description of **error in** and **error out** parameters.

VXIbus Extender VI Descriptions

This section describes VXIbus Extender VIs, which you can use to map specified ECL and TTL trigger lines, VXI utility bus signals, and VXI interrupt levels for the specified mainframe.

MapECLtrig

Maps the specified ECL trigger lines for the specified mainframe in the specified direction (into or out of the mainframe).

mapecl.bmp
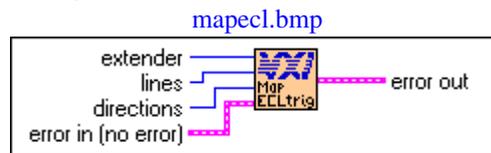
extender specifies the mainframe extender for which to map ECL lines.

lines is a bit vector of ECL trigger lines.
   1:   Enable for appropriate line.
   0:   Disable for appropriate line.

Bits 5 to 0 correspond to ECL lines 5 to 0, respectively.

directions is a bit vector of directions for ECL trigger lines.
   1:   Into the mainframe.
   0:   Out of the mainframe.

Bits 5 to 0 correspond to ECL lines 5 to 0, respectively.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

MapTTLtrig

Maps the specified TTL trigger lines for the specified mainframe in the specified direction (into or out of the mainframe).

mapttl.bmp
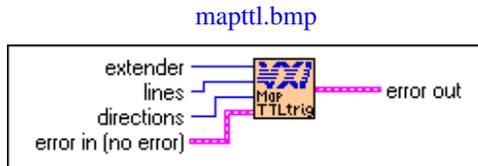
extender specifies the mainframe extender for which to map TTL lines.

lines is a bit vector of TTL trigger lines.

Bits 7 to 0 correspond to TTL lines 7 to 0, respectively.
- 1:   Enable for appropriate line.
- 0:   Disable for appropriate line.

directions is a bit vector of directions for TTL trigger lines.

Bits 7 to 0 correspond to TTL lines 7 to 0, respectively.
- 1:   Into the mainframe.
- 0:   Out of the mainframe.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

MapUtilBus

Maps the specified VXI utility bus signal for the specified mainframe into and out of the mainframe. The utility bus signals include Sysfail, ACfail, and SYSRESET*.

MapUtil.bmp
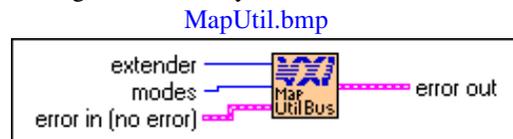
extender specifies the mainframe extender for which to map utility bus signals.

| U16 |

modes is a bit vector of utility bus signals corresponding to the utility bus signals.
   1: Enable for the corresponding signal and direction.
   0: Disable for the corresponding signal and direction.

| Bit	Utility Bus Signal and Direction
5	ACfail into the mainframe
4	ACfail out of the mainframe
3	Sysfail into the mainframe
2	Sysfail out of the mainframe
1	SYSRESET* into the mainframe
0	SYSRESET* out of the mainframe

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

MapVXIint

Maps the specified VXI interrupt levels for the specified mainframe in the specified direction (into or out of the mainframe).

mapvxi.bmp
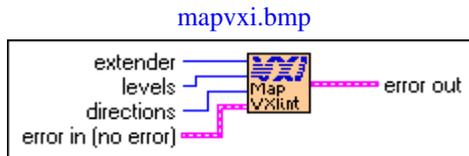
| I16 |

extender specifies the mainframe extender for which to map VXI interrupt levels.

| U16 |

levels is a bit vector of VXI interrupt levels.

Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.
   1: Enable for appropriate level.
   0: Disable for appropriate level.

directions is a bit vector of directions for VXI interrupt levels.

Bits 6 to 0 correspond to VXI interrupt levels 7 to 1, respectively.

    1:   Into the mainframe.
    0:   Out of the mainframe.

error in describes error conditions occurring before the VI executes. Refer to Appendix A, *Error Codes*, for more information about error clusters and specific errors.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise, error out describes the error status of this VI. Refer to Appendix A, *Error Codes*, for specific error information.

Error Codes

This appendix describes the error codes returned by LabVIEW VXI VIs.

In previous versions of LabVIEW, the VXI VIs returned a status value that indicated how the VI completed. The new VXI VIs use LabVIEW error clusters to track status information for the VIs. The error codes used by these VIs are not the same as the previous status values.

In most cases the new error codes have been formed by adding a constant offset to the old status values. The Ecode VI that is used to carry out this conversion is shown below.
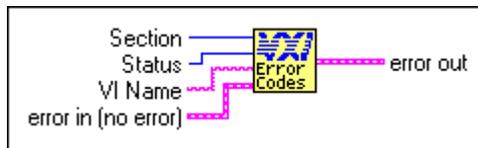
This VI along with the SubVIs it uses are available in the VXI directory for anyone that is interested in how the conversion process is performed.

A new VI, the Oldcode VI, can convert a new error cluster to an old status value. This VI is available in the `Vi.lib\Inst\VXI` directory. It returns the status both as an integer and as a 16-element Boolean array.
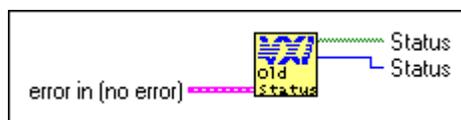
 error in describes error conditions before the VI executes. If an error already occurred, the VI returns the value of the error in cluster in error out.

 Status is a 16-element Boolean array corresponding to the old status value indicated by the error code in the input error cluster.

 Status is an integer error code corresponding to the old status value indicated by the error code in the input error cluster.

Error Cluster Descriptions

The **error in** cluster contains the following information:

status is TRUE if an error occurred. If the status is TRUE the VI does not perform any operations.

code is the error code number identifying an error. A value of 0 means no error, a negative value means a fatal error, and a positive value is a warning/status report. The codes used by the VIs in each chapter are listed below.

source identifies where an error occurred. The source string is usually the name of the VI that produced the error.

error out contains error information. If the error in cluster indicated an error, the error out cluster contains the same information. Otherwise the error out cluster describes the status of this VI.

System Configuration and VXI Library Initialization VIs

–3201	Device not found or device already exists.
–3202	Invalid field or logical address out of range 0 to 511.
–3203	Dynamic memory allocation failure.
–3299	NI-VXI library system error.

Note   The InitVXIlibrary and CloseVXIlibrary VIS do not use these codes. The InitVXIlibrary codes are shown in the following list.

–3001	NI-VXI library initialization failed.
–3099	NI-VXI library system error.
3001	VXI library already initialized (repeat call).
3002	NI-VXI library successfully initialized, but the Resource Manager failed to run successfully.

Note   The CloseVXIlibrary codes are shown in the following list.

–3101	NI-VXI library termination failed.
–3199	NI-VXI library system error.
3101	Success; previous InitVXIlibrary calls still pending.

Word Serial Commander Protocol VIs

–3301	Invalid logical address.
–3302	Invalid abortop.
–3303	Error occurred attempting to open or access the specified file.
–3304	Timeout occurred before command was sent.
–3305	Timeout occurred before response was received.
–3307	Forced user abort occurred during transfer.
–3308	Invalid logical address.
–3309	Multiple query error occurred during transfer.
–3310	Bus error occurred during transfer.
–3311	Timeout occurred during transfer.
–3312	Device did not recognize the command.
–3313	Read protocol error.
–3314	Device reported an input protocol error during transfer.
–3315	Device reported an output protocol error during transfer.
–3316	Violation of raw read protocol occurred during transfer.
–3317	Violation of raw write protocol occurred during transfer.
–3318	Handler timed out.
3303	Transfer completed successfully, and at least one of the specified termination conditions was received.
3305	Transfer completed successfully, and the specified number of bytes were received.
3307	Transfer completed successfully, the specified number of bytes were read, and at least one of the specified termination conditions was received.
3308	Transfer was aborted because the device was not ready.
3319	Transfer completed successfully, and the END bit was received.
3323	Transfer completed successfully, the specified number of bytes were read, and the END bit was received.
3335	Transfer completed successfully, and the termination character was received.

3339	Transfer completed successfully, the specified number of bytes were read, and the termination character was received.
3351	Transfer completed successfully, and both the termination character and the END bit were received.
3355	Transfer completed successfully, the specified number of bytes were read, and both the termination character and the END bit were received.

Word Serial Servant Protocol VIs

–3401	Word Serial servant VI not supported.
–3402	Unable to perform operation (operation in progress).
–3403	Handler timed out.
–3499	NI-VXI library system error.
3401	Operation posted successfully, will begin after WSSenable or **proterr** ignored because previous error was pending.

Low-Level VXIbus Access VIs

–3501	Invalid window number or no hardware support.
–3502	Invalid input parameter.
–3503	Invalid field or address.
–3505	Byte order not supported.
–3506	Specified offset is not accessible from this hardware.
–3507	Privilege not supported.
–3508	Timeout (window still in use; must use UnMapVXIAddress).
–3509	Window does not have owner access.
–3510	Base address change is not supported.
–3599	NI-VXI library system error.
3501	Window accessor was released, but the hardware window is still mapped due to multiple users or byte order set the same for all windows.

✎ **Note**   The VXIpeek and VXIpoke VIs do not use these codes. The codes used by VXIpeek and VXIpoke are shown in the following list.

–3601	Bus error occurred during transfer.
–3699	NI-VXI library system error.

High-Level VXIbus Access VIs

–3601	Bus error occurred during access.
–3602	Invalid access, source or destination parameters.
–3603	Invalid register or address.
–3604	Invalid access width.
–3605	Byte order not supported.
–3606	Specified offset or address is not accessible from this hardware.
–3607	Privilege not supported.
–3608	Timeout; DMA abort error.
–3609	Width not supported.
–3699	NI-VXI library system error.

Local Resource Access VIs

–3701	Unsupported VI; not a slot 0 device or bus error occurred during access.
–3703	Invalid address.
–3704	Invalid access width.
–3709	Access with not supported.
–3799	NI-VXI library system error.

Shared Memory Access VIs

–3801	Memory operation failed.
–3802	Local CPU is A16 only.
–3805	Invalid direction.
–3899	NI-VXI library system error.
3801	Memory allocation successful, but memory must be accessed using VXImemCopy.

VXI Signal VIs

–3901	Invalid logical address or specified signal could not be added/removed due to queue full/empty condition.
–3902	Invalid logical address, or timeout occurred while waiting for a signal to arrive that matches the specified signal mask.
–3903	Handler timed out.
–3999	NI-VXI library system error.
3901	Signal queue full, will enable after a SignalDeq.

VXI Interrupt VIs

–4001	No hardware support for this operation or invalid interrupt level.
–4002	Invalid controller.
–4003	Invalid interrupt level.
–4004	Bus error occurred during IACK cycle.
–4005	VXI interrupt still pending from previous AssertVXIint VI or invalid modes.
–4006	Handler timed out.
–4099	NI-VXI library system error.
4001	Signal queue full, will enable interrupts after a SignalDeq.

VXI Trigger VIs

–4101	Unsupported VI; no hardware support.
–4102	Invalid controller.
–4103	Invalid line or protocol.
–4104	Line not supported.
–4105	Protocol not supported.
–4106	Timeout occurred.
–4107	Line already in use.
–4108	SrcTrig not supported.
–4109	DestTrig not supported.
–4110	Invalid configuration.
–4111	Already mapped, must use UnMapTrigToTrig.

–4112	Line or counter not configured or initialized.
–4115	Previous operation incomplete or previous tick configured and enabled.
–4116	Previous acknowledge still pending.
–4117	No trigger sensed.
–4118	Line not configured for external SEMI_SYNC.
–4119	Handler timed out.
–4199	NI-VXI library system error.

System Interrupt Handler VIs

–4201	No hardware support for this operation or interrupt type.
–4202	Invalid controller.
–4203	Handler timed out.
–4299	NI-VXI library system error.

VXIbus Extender VIs

–4301	Unsupported VI; no hardware support.
–4302	Invalid extender.
–4399	NI-VXI library system error.

B

Technical Support Resources

This appendix describes the comprehensive resources available to you in the Technical Support section of the National Instruments Web site and provides technical support telephone numbers for you to use if you have trouble connecting to our Web site or if you do not have internet access.

NI Web Support

To provide you with immediate answers and solutions 24 hours a day, 365 days a year, National Instruments maintains extensive online technical support resources. They are available to you at no cost, are updated daily, and can be found in the Technical Support section of our Web site at `www.ni.com/support`

Online Problem-Solving and Diagnostic Resources

- **KnowledgeBase**—A searchable database containing thousands of frequently asked questions (FAQs) and their corresponding answers or solutions, including special sections devoted to our newest products. The database is updated daily in response to new customer experiences and feedback.

- **Troubleshooting Wizards**—Step-by-step guides lead you through common problems and answer questions about our entire product line. Wizards include screen shots that illustrate the steps being described and provide detailed information ranging from simple getting started instructions to advanced topics.

- **Product Manuals**—A comprehensive, searchable library of the latest editions of National Instruments hardware and software product manuals.

- **Hardware Reference Database**—A searchable database containing brief hardware descriptions, mechanical drawings, and helpful images of jumper settings and connector pinouts.

- **Application Notes**—A library with more than 100 short papers addressing specific topics such as creating and calling DLLs, developing your own instrument driver software, and porting applications between platforms and operating systems.

Software-Related Resources

- **Instrument Driver Network**—A library with hundreds of instrument drivers for control of standalone instruments via GPIB, VXI, or serial interfaces. You also can submit a request for a particular instrument driver if it does not already appear in the library.

- **Example Programs Database**—A database with numerous, non-shipping example programs for National Instruments programming environments. You can use them to complement the example programs that are already included with National Instruments products.

- **Software Library**—A library with updates and patches to application software, links to the latest versions of driver software for National Instruments hardware products, and utility routines.

Worldwide Support

National Instruments has offices located around the globe. Many branch offices maintain a Web site to provide information on local services. You can access these Web sites from www.ni.com/worldwide

If you have trouble connecting to our Web site, please contact your local National Instruments office or the source from which you purchased your National Instruments product(s) to obtain support.

For telephone support in the United States, dial 512 795 8248. For telephone support outside the United States, contact your local branch office:

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Brazil 011 284 5011, Canada (Calgary) 403 274 9391,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11,
France 01 48 14 24 24, Germany 089 741 31 30, Greece 30 1 42 96 427,
Hong Kong 2645 3186, India 91805275406, Israel 03 6120092,
Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456,
Mexico (D.F.) 5 280 7625, Mexico (Monterrey) 8 357 7695,
Netherlands 0348 433466, New Zealand 09 914 0488,
Norway 32 27 73 00, Poland 0 22 528 94 06, Portugal 351 1 726 9011,
Singapore 2265886, Spain 91 640 0085, Sweden 08 587 895 00,
Switzerland 056 200 51 51, Taiwan 02 2528 7227,
United Kingdom 01635 523545

Glossary

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 the configuration registers of the VXI device. 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.

access   Address modifier codes.

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.

array   Ordered, indexed set of data elements of the same type.

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.
ASIC	Application Specific Integrated Circuit (a custom chip).
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 printed circuit board, 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.
BERR*	Bus Error signal. This signal is asserted by either a slave device or the BTO unit (bus timeout 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.
bit	Binary digit. The smallest possible unit of data: a two-state, yes/no, 0/1 alternative. The building block of binary coding and numbering systems. Several bits make up a *byte*.
bit vector	A string of related bits in which each bit has a specific meaning.
buffer	Temporary memory/storage location for holding data before it can be transmitted elsewhere.
bus master	A device that is capable of requesting the Data Transfer Bus (DTB) for the purpose of accessing a slave device.

bus timeout unit	A VMEbus functional module that times the duration of each data transfer on the Data Transfer Bus (DTB) and terminates the DTB cycle if the duration is excessive. Without the termination capability of this module, a bus master attempt to access a nonexistent slave could result in an indefinitely long wait for a slave response.
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 nsec.
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.
communication 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 moduledevice 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

data transfer bus	One of four buses on the VMEbus backplane. The DTB is used by a bus master to transfer binary data between itself and a slave device.
default handler	Automatically installed at start-up to handle associated interrupt conditions; the software can then replace it with a specified handler.
de-referencing	Accessing the contents of the address location pointed to by a pointer.
DIR	Data In Ready
DIRviol	Data In Ready violation
DOR	Data Out Ready
DORviol	Data Out Ready violation
DRAM	Dynamic RAM (Random Access Memory); storage that the computer must refresh at frequent intervals.

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.
event signal	A 16-bit value written to a Message-Based device's Signal register in which the most significant bit (bit 15) is a 1, designating an Event (as opposed to a Response signal). The VXI specification reserves half of the Event values for definition by the VXI Consortium. The other half are user defined.

Extended Class device	A class of VXIbus device defined for future expansion of the VXIbus specification. These devices have a subclass register within their configuration space that defines the type of extended device.
extended controller	A mainframe extender with additional VXIbus controller capabilities.
Extended Longword Serial Protocol	A form of Word Serial communication in which Commanders Serial Protocol Servants communicate with 48-bit data transfers.
external controller	In this configuration, a plug-in interface board in a computer is connected to the VXI mainframe via one or more VXIbus extended controllers. The computer then exerts overall control over VXIbus system operations.

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.
FIFO	First In-First Out; a method of data storage in which the first element stored is the first one retrieved.

G

GPIB	General Purpose Interface Bus
GPIO	General Purpose Input Output, a module within the National Instruments TIC chip which is used for two purposes. First, GPIOs are used for connecting external signals to the TIC chip for routing/conditioning to the VXIbus trigger lines. Second, GPIOs are used as part of a crosspoint switch matrix.

H

handshaking	A type of protocol that makes it possible for two devices to synchronize operations.
hardware context	The hardware setting for address space, access privilege, and byte ordering.

high-level | Programming with instructions in a notation more familiar to the user than machine code. Each high-level statement corresponds to several low-level machine code instructions and is machine independent, meaning that it is portable across many platforms.

I

IACK | Interrupt Acknowledge

IEEE | Institute of Electrical and Electronic Engineers

IEEE 1014 | The VME specification.

interrupt | A signal indicating that the central processing unit should suspend its current task to service a designated activity.

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.

I/O | Input/output; the techniques, media, or devices used to achieve communication between entities.

L

LabVIEW | Laboratory Virtual Instrument Engineering Workbench

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.

Longword Serial Protocol | A form of Word Serial communication in which Commanders and Servants communicate with 32-bit data transfers instead of 16-bit data transfers as in the normal Word Serial Protocol.

low-level | Programming at the system level with machine-dependent commands.

M

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.
master	A functional part of a MXI/VME/VXIbus device that initiates data transfers on the backplane. A transfer can be either a read or a write.
MB	Megabytes of memory.
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	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.
MQE	Multiple Query Error; a type of Word Serial Protocol error. If a Commander sends two Word Serial queries to a Servant without reading the response to the first query before sending the second query, a MQE is generated.
multitasking	The ability of a computer to perform two or more functions simultaneously without interference from one another. In operating system terms, it is the ability of the operating system to execute multiple applications/processes by time-sharing the available CPU resources.

N

NI-VXI	The National Instruments bus interface software for VME/VXIbus systems.
nodes	Execution elements of a block diagram consisting of functions, structures, and subVIs.
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.

P

peek   To read the contents.

pointer   A data structure that contains an address or other indication of storage location.

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

queue   A group of items waiting to be acted upon by the computer. The arrangement of the items determines their processing priority. Queues are usually accessed in a FIFO fashion.

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.

remote controller   A device in the VXI/VME system that has has no intelligent CPU installed, but it has the capability to control the VXI/VMEbus. An embedded or external controller may use a remote controller to control the remote mainframe.

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.
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.
response signal	Used to report changes in Word Serial communication status between a Servant and its Commander.
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.
ROR	Release On Request; a type of VME bus arbitration where the current VMEbus master relinquishes control of the bus only when another bus master requests the VMEbus.
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.
RRviol	Read Ready protocol violation; a type of Word Serial Protocol error. If a Commander attempts to read a response from the Data Low register when the device is not Read Ready (does not have a response pending), a Read Ready violation may be generated.

S

sec	Seconds
SEMI-SYNC Protocol	A one-line, open collector, multiple-device handshake trigger protocol.
Servant	A device controlled by a Commander.

Shared Memory Protocol	A communications protocol for Message-Based devices that uses a block of memory that is accessible to both a client and a server. The memory block acts as the medium for the protocol transmission.
short integer	Data type of 16 bits, same as *word*.
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.
signed integer	*n* bit pattern, interpreted such that the range is from $-2^{(n-1)}$ to $+2^{(n-1)} - 1$.
slave	A functional part of a MXI/VME/VXIbus device that detects data transfer cycles initiated by a VMEbus master and responds to the transfers when the address specifies one of the device's registers.
SMP	*See* Shared Memory Protocol.
SRQ	Service Request
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.
subVI	VI used in the block diagram of another VI; comparable to a subroutine.
supervisory	One of the defined types of VMEbus data transfers; indicated by certain
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

TIC	Trigger Interface Chip; a proprietary National Instruments ASIC used for direct access to the VXI trigger lines. The TIC contains a 16-bit counter, a dual 5-bit tick timer, and a full crosspoint switch.
tick	The smallest unit of time as measured by an operating system.
top-level VI	VI at the top of the VI hierarchy. This term is used to distinguish the VI from its subVIs.
trigger	A condition for starting or stopping clocks.
tristated	Defines logic that can have one of three states: low, high, and high-impedance.
TTL	Transistor-Transistor Logic.

U

UART	Universal Asynchronous Receiver Transmitter.
unasserted	A signal in its inactive false state.
unsigned integer	n bit pattern interpreted such that the range is from 0 to $2^n - 1$.
UnSupCom	Unsupported Command; a type of Word Serial Protocol error. If a Commander sends a command or query to a Servant which the Servant does not know how to interpret, an Unsupported Command protocol error is generated.

V

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. Called *VICtext* when used on text-based platforms.
virtual instrument (VI)	LabVIEW program; so called because it models the appearance and function of a physical instrument.
VME	Versa Module Eurocard or IEEE 1014.
VMEbus Class device	Also called non-VXIbus or foreign devices when found in VXIbus systems. They lack the configuration registers required to make them VXIbus devices.
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. Called *vxitedit* when used on text-based platforms.

W

wire	Data path between nodes.
word	A data quantity consisting of 16 bits.
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.
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.

write   Copying data to a storage device.

WRviol   Write Ready protocol violation; a type of Word Serial Protocol error. If a
                         Commander attempts to write a command or query to a Servant that is not
                         Write Ready (already has a command or query pending), a Write Ready
                         protocol violation may be generated.