



## Measurement & Automation Explorer Help for NI-VISA™

June 2008, 371395E-01

This help file explains how to use Measurement & Automation Explorer (MAX) for NI-VISA and NI-VXI configuration and programming.

For more information about this help file, refer to the following topics:

[Using Help](#)

[Related Documentation](#)

[Important Information](#)

[Technical Support and Professional Services](#)

To comment on National Instruments documentation, refer to the [National Instruments Web site](#).

© 2005–2008 National Instruments Corporation. All rights reserved.

## **Related Documentation**

The following documents contain information that you might find helpful as you use this help file:

- *NI-VXI Help*
- *NI-VXI API Help*
- *NI-VISA Help*
- *1155-1992 Standard VMEbus Extensions for Instrumentation: VXIbus*
- *GPIB-VXI/C User Manual*

## **Using Help**

[Conventions](#)

[Navigating Help](#)

[Searching Help](#)

[Printing This Help File](#)

## Conventions

This help file uses the following formatting and typographical conventions:

- » 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 tip, which alerts you to advisory information.
-  This icon denotes a note, which alerts you to important information.
-  This icon denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash.
- bold** Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
- dark red** Text in this color denotes a caution.
- green Underlined text in this color denotes a link to a help topic, help file, or Web address.
- italic* Italic text denotes variables, emphasis, cross–references, or an introduction to a key concept. Italic text 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.

## Navigating Help (Windows Only)

To navigate this help file, use the **Contents**, **Index**, and **Search** tabs to the left of this window or use the following toolbar buttons located above the tabs:

- **Hide**—Hides the navigation pane from view.
- **Locate**—Locates the currently displayed topic in the **Contents** tab, allowing you to view related topics.
- **Back**—Displays the previously viewed topic.
- **Forward**—Displays the topic you viewed before clicking the **Back** button.
- **Options**—Displays a list of commands and viewing options for the help file.

## Searching Help (Windows Only)

Use the **Search** tab to the left of this window to locate content in this help file. If you want to search for words in a certain order, such as "related documentation," add quotation marks around the search words as shown in the example. Searching for terms on the **Search** tab allows you to quickly locate specific information and information in topics that are not included on the **Contents** tab.

## Wildcards

You also can search using asterisk (\*) or question mark (?) wildcards. Use the asterisk wildcard to return topics that contain a certain string. For example, a search for "prog\*" lists topics that contain the words "program," "programmatically," "progress," and so on.

Use the question mark wildcard as a substitute for a single character in a search term. For example, "?ext" lists topics that contain the words "next," "text," and so on.



**Note** Wildcard searching will not work on Simplified Chinese, Traditional Chinese, Japanese, and Korean systems.

## **Nested Expressions**

Use nested expressions to combine searches to further refine a search. You can use Boolean expressions and wildcards in a nested expression. For example, "example AND (program OR VI)" lists topics that contain "example program" or "example VI." You cannot nest expressions more than five levels.

## Boolean Expressions

Click the ► button to add Boolean expressions to a search. The following Boolean operators are available:

- **AND** (default)—Returns topics that contain both search terms. You do not need to specify this operator unless you are using nested expressions.
- **OR**—Returns topics that contain either the first or second term.
- **NOT**—Returns topics that contain the first term without the second term.
- **NEAR**—Returns topics that contain both terms within eight words of each other.

# Search Options

Use the following checkboxes on the **Search** tab to customize a search:

- **Search previous results**—Narrows the results from a search that returned too many topics. You must remove the checkmark from this checkbox to search all topics.
- **Match similar words**—Broadens a search to return topics that contain words similar to the search terms. For example, a search for "program" lists topics that include the words "programs," "programming," and so on.
- **Search titles only**—Searches only in the titles of topics.

## Printing Help File Topics (Windows Only)

Complete the following steps to print an entire book from the **Contents** tab:

1. Right-click the book.
2. Select **Print** from the shortcut menu to display the **Print Topics** dialog box.
3. Select the **Print the selected heading and all subtopics** option.  
 **Note** Select **Print the selected topic** if you want to print the single topic you have selected in the **Contents** tab.
4. Click the **OK** button.

## **Printing PDF Documents**

This help file may contain links to PDF documents. To print PDF documents, click the print button located on the Adobe Acrobat Viewer toolbar.

# Glossary

Prefixes	Numbers/Symbols	A	B	C	D	E	F	G	H	I	L
M	N	P	R	S	T	U	V	W			

## Prefixes

Symbol	Prefix	Value
p	pico	$10^{-12}$
n	nano	$10^{-9}$
$\mu$	micro	$10^{-6}$
m	milli	$10^{-3}$
k	kilo	$10^3$
M	mega	$10^6$
G	giga	$10^9$
T	tera	$10^{12}$

## Numbers/Symbols

nV	nanovolts	10 <sup>-9</sup> volts
μV	microvolts	10 <sup>-6</sup> volts
μΩ	microohms	10 <sup>-6</sup> ohms
mΩ	milliohms	10 <sup>-3</sup> ohms
MΩ	megaohms	10 <sup>6</sup> ohms
pA	picoamps	10 <sup>-12</sup> amperes
nA	nanoamps	10 <sup>-9</sup> amperes
μA	microamps	10 <sup>-6</sup> amperes
mA	milliamps	10 <sup>-3</sup> amperes

## A

A16/A24/A32 space VXIbus address spaces. Address space is a set of  $2^n$  memory locations differentiated from other such sets in VXI/VMEbus systems by six signal lines known as *address modifiers*, where  $n$  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.

A16 space is equivalent to the VME 64 KB short address space. In VXI, the upper 16 KB region is referred to as *VXI configuration space*.

A24 space is equivalent to the VME 16 MB standard address space.

A32 space is equivalent to the VME 4 GB extended address space.

## B

- base address    A specified address that is combined with a *relative* address (or offset) to determine the *absolute* address of a data location. All VXI address windows have an associated base address for their assigned VXI address spaces.
- bus master    A type of a plug-in board or controller with the ability to read and write devices on the computer bus.
- byte    A grouping of adjacent binary digits operated on by the computer as a single unit. 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

Commander	<p>A message-based device that is also a bus master and can control one or more <a href="#">Servants</a>. A Commander can itself be a Servant to another Commander that is higher in the <a href="#">Commander/Servant hierarchy</a>.</p>
Commander/Servant hierarchy	<p>The VXIbus specification defines a Commander/Servant communication protocol that you can use to construct hierarchical systems using conceptual layers of VXI devices. The resulting structure is like a tree. A <i>Commander</i> is any message-based device in the hierarchy with one or more associated lower-level devices, or Servants. A <i>Servant</i> is any device in the subtree of a Commander. A device can be both a Commander and a Servant in a multiple-level hierarchy.</p> <p>A Commander has exclusive control of its immediate Servants' (one or more) communication and configuration registers. Any VXI module has one and only one Commander. Commanders use the Word Serial Protocol to communicate with Servants through the Servants' communication registers. Servants communicate with their Commander, responding to the Word Serial commands and queries from their Commander. Servants can also communicate asynchronous status events to their Commander through hardware interrupts, or by writing specific messages directly to their Commander's Signal register.</p>
communication registers	<p>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. VME devices and VXI register-based devices do not have communication registers.</p>
configuration registers	<p>A set of registers through which the system can identify a module device type, model, manufacturer, address space, and memory requirements. To support automatic system and memory configuration, the VXIbus specification requires that all VXIbus devices have a set of such registers. VME devices do not have configuration devices.</p>
configuration space	<p>The VXIbus specification reserves a section of VXI/VMEbus address space for automatic system configuration and base-level communication. This section is the upper 16 KB of what is known as the A16 address space, which is divided into 256 blocks of 64 bytes each for a maximum of 256 VXI devices in a single VXIbus chassis. Therefore, this section (known as VXIbus configuration space) begins at a base address of 0xC000 and ends at 0xFFFF. A unique 8-bit logical address identifies each device. The logical address determine which of these 256 blocks a VXI device resides at or uses. Each VXI device, therefore, has a unique location in the system.</p> <p>You can calculate the offset, or starting address, of a device's 64-byte block of addresses using the following formula:</p> $\text{offset} = \text{C000 hex} + (\text{Logical Address} * 40 \text{ hex})$ <p>or, in decimal:</p> $\text{offset} = 49152 + (\text{Logical Address} * 64)$
controller (System Controller)	<p>A <i>controller</i> is a device that can control other devices. A desktop computer with a MXI interface board, an embedded computer in a VXI chassis, a VXI-MXI, and a VME-MXI may all be controllers depending on the configuration of the VXI system.</p> <p>A <i>VMEbus System Controller</i> is a device configured for installation in Slot 0 of a VXIbus chassis or Slot 1 of a VMEbus chassis. This device is unique in the VMEbus system in that it performs the VMEbus System Controller functions,</p>

including clock sourcing and arbitration for data transfers across the backplane.

A *MXIbus System Controller* is a functional module that has arbiter, daisy-chain driver, and MXIbus cycle timeout responsibility. This device is always the first device in the MXIbus daisy-chain.

## D

- device class VXIbus devices are categorized by their supported protocols into the following four classes.
- Message-based devices support both the VXIbus configuration and communication protocols. These devices have Commander and/or command-based Servant capabilities.
- Register-based devices have VXIbus configuration registers, but do not have communications registers. Typically, they have little or no local intelligence and can be controlled by message-based devices.
- Memory devices have VXIbus configuration registers that are used for either permanent or temporary data storage in blocks of RAM or ROM in VMEbus A24 or A32 address space.
- Extended devices have VXIbus configuration registers and a subclass register, which defines both standard and manufacturer-specific subclasses. The subclass further defines the functionality of an extended device, such as a chassis extender. The National Instruments VXI-MXI chassis extender is an example of an extended device.
- VME devices that do not implement the VXIbus configuration registers do not have a VXI device class.
- DMA Direct Memory Access; a method by which data is transferred between devices and internal memory without intervention of the central processing unit. Generally, DMA is the fastest and most efficient method for transferring data.
- DRAM Dynamic RAM (Random Access Memory); storage that the computer must refresh at frequent intervals.

## E

- embedded controller A computer plugged directly into the backplane of a PXI or VXI chassis. Some examples of embedded controllers are National Instruments PXI-8170 Series and VXIpc controllers.
- extender A device such as the VXI-MXI-2 or VME-MXI-2 that can map interrupt lines, trigger lines, or other signals into or out of a chassis.
- external controller A desktop computer or workstation connected to the VXI system via a MXI interface board. An example is a standard personal computer with a PCI-MXI-2 installed.

## **F**

fair requester An MXIbus master that does not arbitrate for the MXIbus after releasing it until it detects the bus request signal inactive. This ensures that all requesting devices are granted use of the bus.

## G

**GPIB** General Purpose Interface Bus; the industry-standard IEEE 488 bus. The GPIB is a cable bus that connects computers to test equipment. Hewlett-Packard developed the original GPIB in the late 1960s to connect and control their line of programmable instruments (at Hewlett-Packard, the GPIB is called the HP-IB).

In 1975, the Institute of Electrical and Electronic Engineers (IEEE) published ANSI/IEEE Standard 488, IEEE Standard Digital Interface for Programmable Instrumentation, which contained the electrical, mechanical, and functional specifications of an interfacing system. This bus is now used worldwide and is known by the following names:

- General Purpose Interface Bus (GPIB)
- Hewlett-Packard Interface Bus (HP-IB)
- IEEE 488 Bus

The IEEE committee later added a supplemental standard, IEEE 488.2, Codes, Formats, Protocols, and Common Commands, which extended the IEEE 488 specification. IEEE 488.2 defines a bus communication protocol, a common set of data codes and formats, and a generic set of common device commands. It also defines a standard set of instrument commands in the Standard Command for Programmable Instrumentation (SCPI) document.

GPIB became an industry standard for test and measurement systems. Because VXI was developed for this industry, many VXI devices use the IEEE 488.2 and SCPI commands.

**GPIB-VXI/C** Loads the GPIB-VXI code instruments in a debug mode. Debugging a GPIB-VXI/C is described in detail in the *GPIB-VXI/C User Manual*. Normally, you should not place a GPIB-VXI/C into debug mode unless directed to do so by National Instruments technical support.

## **H**

HP/Agilent VISA Refers to the Hewlett-Packard/Agilent implementation of the VISA specification.

## I

instrument driver	Instrument drivers are the tools for developing a system without programming the instruments themselves. By using instrument drivers, you do not need to learn any instrument command sets or data formatting routines. Instrument drivers contain high-level functions for specific instruments such as multimeters, scopes, and counters. Because the drivers have all the programming commands and data formatting routines for instruments, you can concentrate on developing systems rather than programming instruments.
interrupt	A means for a device to notify another device that an event occurred. This asynchronous event suspends normal activity and temporarily diverts activity to an interrupt handling function.
interrupt handler	A functional module that detects interrupt requests generated by interrupters and performs appropriate actions.

## L

LabVIEW	LabVIEW is a graphical programming system designed for easy construction of sophisticated, user-defined instrumentation systems. It has the flexibility necessary to harness the high performance of VXI, along with a programming methodology that dramatically reduces application development time. LabVIEW is fully <i>VXIplug&amp;play</i> compliant to give you the full benefits of <i>VXIplug&amp;play</i> technology.
LabWindows/CVI	LabWindows/CVI is an integrated software system for rapid development, prototyping, and operation of test and measurement and data acquisition applications. It provides a full-function C programming environment for the Windows and Sun Solaris platforms. LabWindows/CVI applications are written in industry-standard program code, and are flexible, modular, extensible, and portable between either platform. National Instruments offers VXI/VME development systems for these two platforms that link the NI-VXI driver software into LabWindows/CVI to control VXI instruments from either embedded VXI/VME controllers or external computers equipped with a MXI interface. LabWindows/CVI is fully <i>VXIplug&amp;play</i> compliant to give you the full benefits of <i>VXIplug&amp;play</i> technology.
logical address	An 8-bit number that uniquely identifies the location of each VXIbus device's configuration registers in a system and indicates Commander and Servant relationships. The A16 register address of a device is $C000h + \text{Logical Address} * 40h$ .
Logical Address 0	<p>If a local device, such as the external controller, is configured at Logical Address 0, it is responsible for the following VXI Resource Manager operations at startup, as defined by the VXIbus specification:</p> <ul style="list-style-type: none"><li>• Identifies all VXIbus devices in the system</li><li>• Manages the system self-tests and diagnostic sequence</li><li>• Configures the system's A24 and A32 address map</li><li>• Configures the system's Commander/Servant hierarchies</li><li>• Allocates the VMEbus IRQ lines</li><li>• Initiates normal system operation</li></ul> <p>The Startup Resource Manager also has the following capabilities, which extend beyond the requirements of the VXIbus specification:</p> <ul style="list-style-type: none"><li>• Multiple chassis support using standard VXIbus chassis extenders (such as the VXI-MXI-2)</li><li>• Support for dynamically configured devices on a per-chassis basis</li><li>• Integration of non-VXI (VME and pseudo-VXI) devices on a per-chassis basis using MAX.</li></ul>
longword	Data type of 32-bit integers. In Longword Serial Protocol, Commanders and Servants communicate with 32-bit data transfers instead of 16-bit transfers as in the standard Word Serial Protocol.

## M

- message-based communication      Message-based devices implement the defined VXIbus registers and communication protocols. These devices are able to use Word Serial Protocol to communicate with one another through communication registers, which are optional registers that register-based devices do not use. Message-based devices communicate at a very high level using ASCII characters, just like GPIB. The ASCII characters that you send to a message-based device must be in the device's specific language, but you do not need to be concerned with module-specific registers (that is, binary reading and writing). Many VXI message-based instruments are also SCPI-compatible.
- MXI/MXI-2      The Multisystem eXtension Interface bus (MXIbus) is a multidrop parallel bus architecture designed for high-speed communication between devices. The MXIbus is a general-purpose gateway that you can use to communicate between two or more devices, such as personal computers, workstation computers, VXIbus chassis, VMEbus-based computers, stand-alone instruments, or modular instruments.
- MXI-2 is the second generation of the National Instruments MXIbus product line. MXI-2 expands the number of signals on a standard MXIbus cable by including VXI triggers, all VXI interrupts, CLK10, SYSFAIL\*, SYSRESET\*, and ACFAIL\*.

## N

NI Spy	The National Instruments utility that tracks an application's calls to NI-VISA, the NI-VXI API, and NI-488.2. You can use this logging utility to obtain a list of the functions the applications calls and quickly find which ones did not execute properly.
NI-VISA	NI-VISA is the native API for communicating with VXI/VME devices. NI-VISA is the National Instruments implementation of the VISA I/O standard, which is a common interface to many types of instruments (such as VXI, GPIB, PXI, serial, TCP/IP, etc.). NI-VXI is optimized for use through NI-VISA, and National Instruments recommends using NI-VISA to develop all new VXI/VME applications.
NI-VXI	NI-VXI is the software package that ships with National Instruments VXI controllers. NI-VXI includes Measurement & Automation Explorer (MAX), NI-VISA, NI Spy, Resource Manager (Resman), VXI device drivers, and other utilities for configuring and controlling your VXI system.
NI-VXI API	The NI-VXI API is an optional development environment that is not part of the default NI-VXI installation. The NI-VXI API was developed before NI-VISA; while NI-VXI still supports the NI-VXI API, National Instruments recommends using NI-VISA for all new VXI/VME applications. If you must develop an application using the older NI-VXI API, run the NI-VXI installer and select the appropriate option in the custom installation screen.
non-Slot 0	Any slot in a VXI chassis except for the first slot.  <b>Caution</b> Installing a device configured as a non-Slot 0 device into Slot 0 can damage the device, the VXIbus backplane, or both.
nonprivileged access	One of the defined types of VMEbus data transfers; indicated by certain address modifier codes. Each defined VMEbus address space has a defined nonprivileged access mode.

## P

- peek To read a single [byte](#), [word](#), or [longword](#) from a particular address. This can be accessed either through a direct dereference of a pointer or through the NI-VISA function `viPeekX` or NI-VXI API function `VXIpeek`.
- pointer A data structure that contains an address or other indication of storage location.
- poke To write a single [byte](#), [word](#), or [longword](#) to a particular address. This can be accessed either through a direct dereference of a pointer or through the NI-VISA function `viPokeX` or NI-VXI API function `VXIpoke`.
- pseudo logical address A number signifying an address that you can use to integrate VME devices with VXI devices. Because the Resource Manager does not configure VME devices, you must manually add the devices. You can choose a number in the range of 256 to 511 (255 and below are reserved for VXI devices). Enter other appropriate information into the various fields of the editor, and when you run Resource Manager, it can then properly configure the various device-specific VME address spaces and VME interrupt lines.

## R

**register-based communication** A register-based device is a Servant-only device that supports VXIbus configuration registers. These devices do not implement the optional communication registers that are required for message-based communication. Register-based devices are typically controlled by message-based devices via device-dependent register reads and writes. The obvious advantage of this is speed; register-based devices communicate literally at the level of direct hardware manipulation. There is no command string parsing overhead. The disadvantage is that each device is different and requires customized manipulation of registers. Thus, the instrument interface is not portable across instruments.

**Resource Manager (Resman)** A message-based Commander, located at Logical Address 0, that provides configuration management services such as address map configuration, Commander and Servant hierarchy mappings, and self-test and diagnostic management. On system startup, the VXI System Controller executes the Resource Manager to automatically configure the entire system. This automatic startup configuration eases system integration. After the Resource Manager has executed, the system is ready to go. You can also execute the Resource Manager interactively at any time if you are reconfiguring system resources.

## S

Servant	A device controlled by a Commander in the <a href="#">Commander/Servant Hierarchy</a> . A Servant can itself be a Commander to Servant devices that are lower in the hierarchy.
shared memory	A block of memory that is accessible to both a client and a server. The memory block operates as a message buffer for communications.
Slot 0	The first slot in a VXI chassis. A device configured for installation in this slot is unique in the VXI system in that it performs the VMEbus System Controller functions, including clock sourcing and arbitration for data transfers across the backplane.  <b>Caution</b> Installing a device configured for Slot 0 into any other slot can damage the device, the VXI backplane, or both.
soft front panel (SFP)	See <a href="#">VXI plug&amp;play soft front panel</a> .
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.
supervisory access	One of the defined types of VMEbus data transfers; indicated by certain address modifier codes.
SYNC protocol	The most basic trigger protocol, simply a pulse of a minimum duration on any one of the trigger lines.  You can assert a pulse for a minimum of 30 ns, followed by a minimum nonassertion period of 50 ns. There is no acknowledgment from the receiver(s). Any module can issue the triggering pulse, but the module cannot be sure that the pulse has been received.
system registry	The system registry is used to store persistent information by both the operating system and many programs. See your operating system documentation for more information.

## **T**

trigger Either TTL or ECL lines used for intermodule communication.

## U

user window A region of PCI address space reserved by the external controller or VXIpc series embedded controller for use via low-level function calls. The `MapVXIAddress()` and `viMapAddress()` functions use this address space to allocate regions for use by the `VXIpeek()/VXIpoke()` and `viPeekXX()/viPokeXX()` functions.

## V

**VIC/VICtext** VXI Interactive Control program, a part of the NI-VXI API bus interface software. Used to program VXI devices, and develop and debug VXI application programs. It is also very useful for developing an understanding of how you can use the NI-VXI API to interact with your VXI devices. Called *VICtext* when used on text-based platforms.



**Note** The VIC utility is not part of the basic NI-VXI installation. However, if you want to use VIC, run the NI-VXI installer and select a "custom" installation instead of "typical". In the **Select Features** dialog box, enable the **NI-VXI API Development** option and continue as prompted. After you reboot, VIC is ready for use.

**VISA** VISA is a multivendor I/O software standard approved by the *VXIplug&play* Systems Alliance. It provides a common foundation for the development, delivery, and interoperability of high-level multivendor system software components, such as instrument drivers, soft front panels, and application software.

NI-VISA is the National Instruments VISA solution.

**VISAIC** VISA Interactive Control (VISAIC) utility, a part of the NI-VISA interface software. You can use this utility to interact with devices using NI-VISA. This is very useful for developing and debugging VISA application programs. It is also very useful for developing an understanding of how you can use VISA to interact with your devices.

**VME** Versa Module Eurocard or IEEE 1014.

**VME64 protocol** Devices that support VME64 protocol can transfer data in 64-bit increments to double traditional VXI throughput to 80 Mbytes/s.

**VXI** VMEbus Extensions for Instrumentation. Introduced by the VXIbus Consortium in 1987, the VXIbus (IEEE 1155) standard combines the best technology from GPIB instruments with modern computer bus architecture—VERSAbus Module Eurocard (VME). VXI uses a chassis with modular instruments on plug-in boards. Due to its VMEbus background, VXI features higher performance and more precise timing and synchronization between instruments. It is also smaller than GPIB rack-and-stack instruments. Yet unlike VME, VXI defines a standard communication protocol to certain devices. Through this interface, you can use common ASCII commands to control the instruments, just as with GPIB. The VXIbus specification is an extension of the VMEbus (IEEE 1014) specification. As an electromechanical superset of the VMEbus, the VXIbus uses the same backplane connectors as VME, the same board sizes, and the same signals defined in the VMEbus specification. The VXIbus adds two board sizes, changes module width, and defines additional signals on the backplane.

**VXI embedded controller** A computer plugged directly into the backplane of a VXI chassis. An example is the National Instruments VXIpc series.

**VXI 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.

**VXI trigger** VXI triggers are a backplane feature that VXI added to the VME standard to achieve the required timing and signaling propagation between controllers and/or instruments. Every

VXI board with a P2 connector has access to eight 10 MHz TTL trigger lines; if it has a P3 connector, it also has access to six 100 MHz ECL trigger lines. The VXIbus specification defines several trigger protocols that your application can use for device synchronization, for stepping through tests, or for a command path. The most basic protocols are SYNC, ASYNC, SEMI-SYNC, START/STOP, and ON/OFF.

*VXIplug&play* soft front panel *VXIplug&play*-compliant instruments have software that can run without a programming environment and can produce a soft front panel (SFP). The SFP is a test application that represents and controls the instrument functions in an interactive way. They use a graphical interface and a mouse pointing device to manipulate simulated knobs, buttons, controls, and displays. You can use the SFP to check the operation of the instrument, verify and correct configuration and installation, and interact with the instrument.

## W

- window All accesses to the VXI/VMEbus address spaces are performed by reads and writes to particular offsets within the local CPU address space, which are made to correspond to addresses on the VXI/VMEbus (using a hardware interface). *Windows* are the areas where the address space of the local CPU is mapped onto the VXI/VMEbus. The sizes and numbers of windows present vary depending on the hardware being used.
- word The standard number of bits that a processor or memory manipulates at one time. Microprocessors typically use 8-, 16-, or 32-bit words. Standard Word Serial Protocol defines a word as 16 bits.
- word serial protocol The simplest required communication protocol supported by message-based devices in the VXIbus system. It uses the A16 communication registers to perform 16-bit data transfers using a simple polling handshake method. All message-based devices are required to support this protocol. Word Serial Protocol transfers data messages to and from devices serially, one **byte** (or *word*) at a time through the device's communication registers.

## **Getting Started with Your Help**

[How is this help file organized?](#)

[What should I do if I can't find the answer to my question here?](#)

[How do I use the Troubleshooter?](#)

## **How is this help file organized?**

The topics are arranged in three major categories:

[How Do I...?](#)

[Why Can't I...?](#)

[Why Would I/When do I...?](#)

Find the category and topic using the Contents or Index tabs. You can also do a full key word search using the Search tab.

## **What should I do if I can't find the answer to my question here?**

You can find more detailed information in the manuals that ship with your kit. Many of these manuals are available as Acrobat (.pdf) files or compiled HTML help (.chm) files, and are contained in the software's manuals directory. NI-VISA includes online help to help you get started with basic concepts about programming with NI-VISA, including programmer's reference with descriptions of each function.

If the documentation does not answer your question, consult the [technical support resources](#) on the National Instruments Web site.

## How do I use the Troubleshooter?

When the device is displayed with an error or a warning, the **Troubleshoot** button in the **Device Properties** dialog is enabled. Click the button to get help with your problem.

## **How Do I...?**

[General Topics](#)

[Configuration](#)

[General Programming Issues](#)

## **General Topics**

[Obtain specifications on my National Instruments hardware](#)

[Obtain the latest National Instruments VISA and VXI software and bug fixes](#)

[Obtain the latest National Instruments product information](#)

[Access the National Instruments Product Knowledgebase](#)

[Debug a VXI problem](#)

[Find out if my device has an instrument driver](#)

[Find out more about programming in NI-DAQ](#)

[Find out more about LabVIEW](#)

[Find out more about LabWindows/CVI](#)

[Find out more about GPIB](#)

## **Obtain specifications on my National Instruments hardware**

You can find most of the specifications for National Instruments VXI hardware products in the Specifications appendix of our getting started manuals or user manuals. If you need information that is not documented in your manual set, see our [technical support Web pages](#). You can also find many specifications for National Instruments data acquisition products in our catalogue.

## **Obtain the latest National Instruments VISA and VXI software and bug fixes**

You can find the latest versions of most NI-VISA and NI-VXI software on the National Instruments [technical support Web pages](#), including [ni.com/downloads](http://ni.com/downloads). On these pages, you can find updates, bug fixes, and patches to your National Instruments software. If you need any of this software on CD-ROM, contact your sales representative or call National Instruments.

## **Obtain the latest National Instruments product information**

The [National Instruments Web site](#) contains the latest information about National Instruments products as well as our online catalogue. We also publish a printed catalogue every year. If you do not have a catalogue and would like to receive one, contact your sales representative or call National Instruments.

## **Access the National Instruments KnowledgeBase**

You can access the KnowledgeBase from the National Instruments [technical support Web pages](#). This database contains information on problems, bugs, questions, and their resolutions that many of our customers have experienced.

## Debug a VXI problem

Large VXI systems can be very complex, so your first task in debugging a VXI problem is to narrow the problem to as few variables as possible. For example, if you have a multichassis system, you may want to try to reproduce the problem with a single chassis. You may even need to pull instruments out of the system until you can narrow the problem to one or two instruments.

[VISAIC](#) and [VIC](#) are two very useful tools for isolating a problem.

Reproducing a VXI problem using these utilities usually eliminates your application as the source of the problem. Using these interactive utilities, you can check basic communication with any instrument with the most basic interface to the driver.



**Note** The VIC utility is not part of the basic NI-VXI installation. However, if you want to use VIC, run the NI-VXI installer and select a "custom" installation instead of "typical." In the **Select Features** dialog box, enable the **NI-VXI API Development** option and continue as prompted. After you reboot, VIC is ready for use.

If you cannot reproduce the problem with VISAIC or VIC, use [NI Spy](#). NI Spy is a powerful utility that can track every NI-VISA, NI-VXI, or NI-488.2 call you make. It highlights calls that return errors or warnings and displays all parameters sent to the driver so you can check their validity. NI Spy can help you locate bugs quickly in large applications.

See also [Debug a VXI program](#).

## **Find out if my device has an instrument driver**

If your instrument driver is already installed, the utilities VISAIC and MAX may be able to find them. [VISAIC](#) and MAX try to find the instrument drivers on your system and launch their soft front panels.

Many instrument drivers are available on the National Instruments Web site. See the [technical support Web pages](#) for LabVIEW and LabWindows™/CVI™ instrument drivers.

If National Instruments does not have an instrument driver for your device, contact the manufacturer. Vendors who are members of the *VXIplug&play* Systems Alliance create many instrument drivers for their products.

## **Find out more about programming in NI-DAQ**

Many example programs are available when you install NI-DAQ. These programs are excellent references for learning how to program National Instruments data acquisition devices. In addition, you can use the DAQ Solution Wizard, which helps you find examples in LabVIEW that are relevant to your application. If you want more formal training, National Instruments offers several different training classes on most of our products, including NI-DAQ. Contact your sales representative or see our [technical support Web pages](#) for more information.

## **Find out more about LabVIEW**

[LabVIEW](#) includes many example programs that are excellent references for learning LabVIEW programming. Many LabVIEW programming books are also available. If you want more formal training, National Instruments offers training courses on many of our products including LabVIEW. Contact your sales representative or see our [technical support Web pages](#) for more information.

## **Find out more about LabWindows/CVI**

[LabWindows/CVI](#) includes many example programs that are excellent references for learning LabWindows/CVI programming. Contact your sales representative or see our [technical support Web pages](#) for more information.

## **Find out more about GPIB**

NI-488.2 includes many example programs that are excellent references for learning NI-488.2 programming. Contact your sales representative or see our [technical support Web pages](#) for more information.

## **Configuration**

[Assign an alias to my VISA device](#)

[Disable a VISA device](#)

[Verify my system is working properly](#)

[Install VXIplug&play software](#)

[Configure NI-VISA to operate with a Hewlett-Packard GPIB-VXI board](#)

[Configure NI-VISA to operate with a National Instruments GPIB-VXI/C](#)

[Configure NI-VISA to see multiple GPIB-VXI/C boards](#)

[Use VISAIC/VIC](#)

[Determine what version of software I have installed](#)

## **Assign an alias to my VISA device**

You can assign devices that support aliases by right-clicking on the device and selecting **Rename** from the pop-up menu.

## **Disable a VISA device**

You can disable a VISA device by going to the **General** tab in the view and unchecking the **Device enabled** box.

## Verify my system is working properly

If your device is not present in the system view in MAX, NI-VISA did not detect it. Remember that you must add the following types of devices to the system manually, using the Add Device wizard:

- All TCP/IP resources
- All Ethernet-based converters (for example, GPIB-ENET and Serial ENET)
- All VME resources
- GPIB-VXI controllers at locations other than primary address 1, secondary address 0

For VXI systems, you may also need to run the Resource Manager if you do not have it configured to run automatically.

If NI-VISA or the VXI Resource Manager detected any errors, the device icons are marked accordingly. To find out what is wrong, go to these marked devices and view the **Troubleshooting** information in the **General** tab.

After you are satisfied that MAX detected all your devices, you can test access to them. The Open VISA Test Panel command creates a session in the VISA Interactive Control and allows you to establish basic communication with your devices. How you test communication with a device varies. You can read the registers of VXI or PXI devices to establish contact. Many GPIB, VXI, and Serial devices respond to the \*IDN? string, which you can send to a device using viWrite(). This 488.2 command queries for the identification string of a device that is returned through viRead().

See also [See my GPIB devices](#) and [See my VXI devices](#).

## Install *VXIplug&play* software

*VXIplug&play* instrument drivers use a standard installer under Windows.

Use the File Manager or Windows Explorer to locate the setup.exe installation program for the instrument driver and double-click it. This runs the *VXIplug&play* installation program to install your driver files in their proper locations.

Typically, *VXIplug&play* instrument drivers install C source code, Windows Help files, a Win32 DLL, LabWindows/CVI Function Panels, KnowledgeBase files, and a stand-alone executable called a [Soft Front Panel](#).

The source code, help files, and Soft Front Panel are installed in the <Drive>\VXIPNP\<Operating System>\<Instrument Name>\ path (for example, C:\VXIPNP\WinNT\HPE-1431).

The DLLs are installed in <Drive>\VXIPNP\<Operating System>\bin\.

You can use the Soft Front Panel to quickly get started using your instrument.

LabWindows/CVI Function Panel files create a function panel tree in LabWindows/CVI. This useful structure contains a logical tree of the functions available in the DLL along with built-in help.

Of course, there are many functions built into the DLL that need to be formed into the final application.

A KnowledgeBase file is a hardware database containing useful specifications for an instrument. These might include operating specifications and capabilities.

In addition, the instrument driver may come with other files to help users program their instruments such as resource files or BASIC source code.

Finally, you may easily port a text-based instrument driver to LabVIEW. Simply use the **Update *VXIplug&play* Drivers** option under the **File** menu in LabVIEW. This checks your VXIPNP directory for installed *VXIplug&play* instrument drivers and converts them into a LabVIEW library containing all the functions listed in the LabWindows/CVI Function Panels. The Function Panels are converted to LabVIEW front panels, while each VI contains a Call Library Function to the DLL. The converted

VIs are placed in your <Driver>\LabVIEW\Instr.lib directory. Under the **Functions** menu, these same VIs appear in the **Instrument Library** subpalette, ready for you to create an instrument driver. Notice that **Convert CVI FP File** under the **File** menu performs the same operations, except that the user must explicitly search for the DLL and FP files.

## **Configure NI-VISA to operate with an HP/Agilent GPIB-VXI board**

To configure NI-VISA to operate with a non-National Instruments GPIB-VXI board, such as the HP1406, you need to add this device to your system manually in MAX. You can do this by right-clicking on **Devices and Interfaces** in the configuration tree and selecting **Create New....** In the wizard, follow the instructions for entering the GPIB address (primary and secondary), VXI logical address, GPIB controller number (which GPIB plug-in board the GPIB-VXI/C is connected to), and the GPIB-VXI number. The GPIB-VXI board number should be unique in your system.

You should also make sure that the appropriate GPIB-VXI module for VISA is installed in your system. Contact the vendor of your GPIB-VXI controller for more information.

## **Configure NI-VISA to operate with a National Instruments GPIB-VXI/C**

By default, the National Instruments GPIB-VXI/C is configured for primary address 1, secondary address 0. NI-VISA automatically detects it at this location. To use the GPIB-VXI/C with a different GPIB address, you need to add this device to your system manually in MAX. You can do this by right-clicking on **Devices and Interfaces** in the configuration tree and selecting **Create New...** In the wizard, follow the instructions for entering the GPIB address (primary and secondary), VXI logical address, GPIB controller number (which GPIB plug in board the GPIB-VXI/C is connected to), and the GPIB-VXI number. The GPIB-VXI board number should be unique in your system.

If you have multiple GPIB-VXI controllers in your system, you need to configure only the ones that are not at the default GPIB address of primary address 1, secondary address 0.

## **Configure NI-VISA to see multiple GPIB-VXI/C boards**

By default, the National Instruments GPIB-VXI/C is configured for primary address 1, secondary address 0. NI-VISA automatically detects it at this location. To use the GPIB-VXI/C with a different GPIB address, you need to add this device to your system manually in MAX. You can do this by right-clicking on **Devices and Interfaces** in the configuration tree and selecting **Create New...** In the wizard, follow the instructions for entering the GPIB address (primary and secondary), VXI logical address, GPIB controller number (which GPIB plug-in board the GPIB-VXI/C is connected to), and the GPIB-VXI number. The GPIB-VXI board number should be unique in your system.

If you have multiple GPIB-VXI controllers in your system, you need to configure only the ones that are not at the default GPIB address of primary address 1, secondary address 0.

## Use VISAIC/VIC

You can launch VISAIC from the **Start»Programs»National Instruments»VISA** menu. You can launch VIC from the **Start»Programs»National Instruments»VXI»NI-VXI API** menu.



**Note** The VIC utility is not part of the basic NI-VXI installation. If you want to use VIC, run the NI-VXI installer and select a "custom" installation instead of "typical." In the **Select Features** dialog box, enable the **NI-VXI API Development** option and continue as prompted. After you reboot, VIC is ready for use.

You can use these two programs to communicate with your devices interactively. You can use VISAIC to open a session to a VXI, GPIB, PXI, TCP/IP, or serial device. Once you open a session to the device, you can send and receive messages, view device attributes, and—if it is a VXI device—directly access the device's memory and registers.

VIC uses NI-VXI API functions to access the VXI bus. You can send and receive word serial messages using the `WSwrt` and `WSrd` functions, access device registers using `VXIinReg` and `VXIoutReg`, and directly access VXI memory space using `VXIin` and `VXIout`.

Both of these programs serve not only to establish initial communication with your devices, but also to verify that the NI-VISA and NI-VXI API libraries are correctly installed and functioning. If you cannot access a device with these utilities, chances are you cannot access it from your application. Once you determine how to access a device with VISAIC or VIC, you can use those same functions in your program.

## Determine what version of software I have installed

In MAX, select **System Information** from the **Help** menu. Click on the **System** tab to display version information on currently installed National Instruments components including NI-VISA, NI-VXI, and T&M utilities.

You can also expand the **Software** category underneath **My System** in MAX and select a component to view its version in the right pane of MAX.

From VISAIC, select the **About...** option from the **Help** menu. This shows the version of NI-VISA that you are using, and also what version of the VISA specification it conforms to.

In addition, all of the 32-bit driver components have version resources. Under Windows Explorer, you can right-click any component and select the **Properties** option. This displays a property sheet with a **Version** tab. This tab has version information about the software and particular components.

You can also run the NI-VXI API utility program VIC and type ver at the prompt. The utility displays the versions of VIC and NI-VXI, and the latest board revision that this NI-VXI driver supports.

## **General Programming Issues**

[Handle VXI Interrupts \(signals, triggers, and so on\) using NI-VISA or the NI-VXI API](#)

[Communicate with register-based devices in NI-VISA or the NI-VXI API](#)

[Communicate with message-based devices in NI-VISA or the NI-VXI API](#)

[Convert a GPIB-VXI/C program from NI-488.2 to NI-VISA](#)

[Program a GPIB-VXI/C with VISA](#)

[Move a large block of data in NI-VISA or the NI-VXI API](#)

[Transfer data using VME64](#)

[Debug a VXI program](#)

[Write a VXI program in C](#)

[Write a VXI program in LabVIEW](#)

[Write a VXI program in Visual Basic](#)

## Handle VXI Interrupts (signals, triggers, and so on) using NI-VISA or the NI-VXI API

### NI-VISA

NI-VISA provides two mechanisms to allow for flexible event handling: queuing and callbacks. The most common way to handle events is with the event queue. After opening a VISA session with `viOpen()`, you can enable the queue by calling `viEnableEvent()` and specifying `VI_QUEUE` as the mechanism. Later, you call `viWaitOnEvent()` to wait for and receive an event from the device associated with that session. If you prefer callbacks, first specify a callback handler with `viInstallHandler()` and then call `viEnableEvent()` with `VI_HNDLR` as the mechanism. The callback handler is invoked whenever the specified event type is received.

The available event types for the VXI interface include:

- `VI_EVENT_VXI_SIGP`—for signals or 16-bit interrupts from VXI devices
- `VI_EVENT_VXI_VME_INTR`—for 8-bit, 16-bit, or 32-bit interrupts from VXI or VME devices
- `VI_EVENT_TRIG`—for sensing triggers from VXI devices

Most VXI devices generate 16-bit interrupts, so you can use either `VI_EVENT_VXI_SIGP` or `VI_EVENT_VXI_VME_INTR`—these are functionally equivalent. If a VXI device generates triggers, you must first specify in `VI_ATTR_TRIG_ID` the TTL or ECL trigger line to sense before calling `viEnableEvent()`.

## NI-VXI API



**Note** National Instruments recommends using NI-VISA, rather than the NI-VXI API, to develop your applications.

The most common way to handle asynchronous signals in the NI-VXI API is to use callback functions. These functions are invoked whenever a specified asynchronous signal is received. After initializing the library with `InitVXIlibrary()`, you can set up a callback function using one of the following functions:

- `SetVXIintHandler()`—to handle VXI interrupts
- `SetSignalHandler()`—to handle VXI signals
- `SetTrigHandler()`—to handle VXI triggers

Then you must also enable the specific signal you are trying to detect using the following functions:

- `EnableVXIint()`—to enable VXI interrupt lines
- `EnableSignalInt()`—to enable VXI signals
- `EnableTrigSense()`—to enable triggers

VXI [interrupts](#) and signals also require that you perform some routing. See the functions `RouteSignal()` and `RouteVXIint()` for more information. You should also look at the interrupt example in the NIVXI directory.

To use VXI triggers, read National Instruments Technical Note 40, *Triggering with NI-VXI*. You can find this document on the National Instruments [technical support resources](#) Web pages.

## **Communicate with register-based devices in NI-VISA or the NI-VXI API**

Register-based devices are required to implement only the four basic VXIbus configuration registers, but most of these devices implement additional instrument-specific registers. There is no standard communication protocol for controlling register-based devices.

Each manufacturer must specify the pattern of register accesses that controls the specific functions of each device. Controllers communicate with register-based devices through register reads and writes in exactly the same way controllers communicate with VMEbus devices.

We have written examples of how to access VXI/VME address space directly using NI-VISA or the NI-VXI API. (National Instruments recommends using NI-VISA, rather than the NI-VXI API, to develop your application.) The VISA register-level communication examples, `HighReg.c` and `LowReg.c`, are in your `vxipnp\win32\NIvisa\examples` directory. The NI-VXI API examples are called `VXIlow.c` and `VXIhigh.c`. These files are in the `NIVXI` directory in the `examples` folder. `VXIlow.c` shows how to use `MapVXIAddress()` and `VXIpeek()/VXIpoke()`. `VXIhigh.c` uses the high-level calls, such as `VXIin` and `VXIout`.

## **Communicate with message-based devices in NI-VISA or the NI-VXI API**

Message-based devices implement a second set of registers in addition to the VXIbus configuration registers. These registers, called VXIbus communication registers, are located within the 64-byte device configuration space. Word Serial Protocol, the defined communication protocol for message-based devices that is similar to the IEEE 488 protocol, is based on these VXIbus communication registers. All message-based devices, regardless of manufacturer, must follow the same procedure when using Word Serial Protocol for communication.

You can communicate with your VXI message-based devices using NI-VISA with the `viRead()` and `viWrite()` functions. Once you have opened a session to the message-based device that you want to communicate with, you can write message strings to it using `viWrite()` and read messages back from the device with `viRead()`. There is an example of how to do message-based communication in the VISA examples directory called `RdWrt.c`.

You can also use the NI-VXI API Word Serial functions. (National Instruments recommends using NI-VISA, rather than the NI-VXI API, to develop your application.) `Wswrt()` is the function you use to send a message string to a device with a particular logical address. `Wsrdr()` is the function you use to receive a message from a device. An example of how to do message-based communication with the NI-VXI API is in the examples folder in the NI-VXI directory. Open the file called `VXIws.c` in Notepad.

## Convert a GPIB-VXI/C program from NI-488.2 to NI-VISA

For controlling message-based VXI devices through a GPIB-VXI, the biggest difference between a program using NI-488 and one using NI-VISA is in the calls made at the beginning and the end. For register-based devices, the differences are more significant. This topic first discusses the basic changes common to both types of devices, then discusses some of the changes required for register-based programming.

For message-based programming, an NI-488 program would typically call `ibdev()` with the VXI device's primary and secondary GPIB addresses to get a handle to the specific device. In NI-VISA, a program calls `viOpen()` with the VXI device's logical address (which is a more natural address because the device is VXI) to get a handle to it. The calls to read and write blocks of data, as well as other IEEE 488 communications (such as reading status bytes and clearing buffers), have a one-to-one mapping between NI-488 and NI-VISA. One difference to be aware of is that when waiting for service requests, a VISA application must first be enabled for that event by using `viEnableEvent()`. At the end of the application, use `viClose()` instead of `ibonl` to close the device handle.

Register-based programming does not have a straightforward mapping. Because register accesses using the GPIB-VXI involve sending requests to the controller itself (using the local command set), NI-488 programs would use `ibdev()` with the GPIB-VXI controller's primary and secondary GPIB addresses. In NI-VISA, you call `viOpen()` with the VXI device's logical address—this is the same for both message-based and register-based devices and is a more natural and intuitive API—and VISA handles sending the necessary messages to the controller. For programming the device, the following NI-488 messages and NI-VISA operations are roughly equivalent:

NI-488	NI-VISA
"Laddr?" or "DLAD?"	<code>viFindRsrc()</code>
"RmEntry?" or "DINF?"	<code>viGetAttribute()</code>
"Cmdr?"	<code>viGetAttribute()</code> with <code>VI_ATTR_CMDR_LA</code>
"LaSaddr?"	<code>viGetAttribute()</code> with <code>VI_ATTR_GPIB_SECONDARY_ADDR</code>
"Primary?"	<code>viGetAttribute()</code> with <code>VI_ATTR_GPIB_PRIMARY_ADDR</code>

"WREG" or "A16"	viOut16() with VI_A16_SPACE
"RREG?" or "A16?"	viIn16() with VI_A16_SPACE
"A24"	viOut16() with VI_A24_SPACE
"A24?"	viIn16() with VI_A24_SPACE
"SrcTrig"	viAssertTrigger()

Notice that with the INSTR register access operations `viOut16()` and `viIn16()`, you pass a device-relative offset in the specified address space. This is different from the GPIB-VXI's local command set, which accepts absolute addresses. If your application currently uses absolute addressing and you do not want to convert to device-relative offsets, you may consider the MEMACC resource, which accepts absolute addressing. You can use the operations `viOut8()` and `viIn8()` to perform 8-bit accesses, which is not a feature supported by the local command set. VISA also defines 32-bit operations and accesses to A32 space, but because these are not implemented by the GPIB-VXI itself, they return errors.

If you have used the DMAmove code instrument, you can now use the `viMoveInxx()` and `viMoveOutxx()` operations instead. They make use of the GPIB-VXI's DMA functionality, but require only a single operation call, instead of the multiple calls required to send the command and data blocks and then poll waiting for the operation to complete. Using VISA to move blocks of data also means that you no longer need to load the DMAmove code instrument, as NI-VISA automatically downloads a separate code instrument to handle these and other operations.

In summary, using NI-VISA to program VXI devices controlled by a GPIB-VXI is no different than if they are controlled with a native VXI controller such as an external controller or the VXIpc series. Although porting the code from NI-488 to NI-VISA is not simple in the case of register-based programming, the easier API and the compatibility with native VXI controllers should be worth the change.

## **Convert an NI-VXI API program to NI-VISA**

For many applications controlling VXI devices, the biggest difference between a program using the NI-VXI API and one using NI-VISA is in the calls made at the beginning and the end. The most significant differences between the two APIs affect programs that route and handle events such as VXI triggers, interrupts, and signals. This topic first discusses the basic changes common to any NI-VXI API program, then discusses some of the specifics for register-based and event-handling programs.

One good resource for converting NI-VXI API programs to NI-VISA is the NI-VXI help, which includes a section about each group of VXI operations, sorted by functionality. (For example, one section discusses high-level register-based programming.) At the beginning of each group overview is a table listing the NI-VXI API and NI-VISA functions associated with that group's operations. You can use the table to match any NI-VXI API function with its NI-VISA equivalent.

## **Starting Your Application**

Typically, an NI-VXI API program calls `InitVXIlibrary()` to begin the program, and then perhaps `FindDevLA()` to get the logical address of a specific device to begin communication. In NI-VISA, a program calls `viOpenDefaultRM()` to initialize NI-VISA, then `viFindRsrc()` to find the device and `viOpen()` to get a handle to it.

## Performing VXI I/O

For a message-based VXI device, the calls to read and write blocks of data, as well as other message-based communications, have a simple mapping between NI-VXI and NI-VISA: `WSwrt()` becomes `viWrite()`, `WSrd()` becomes `viRead()`, etc. For Word Serial Servant functionality, use the `SERVANT` resource in NI-VISA.

Likewise, register-based I/O is performed using NI-VISA functions that are very similar to their NI-VXI API counterparts: `viInX()` instead of `VXIin()`, `viMoveX()` instead of `VXImove()`, etc. In general, NI-VISA attributes control access parameters from the NI-VXI API. Note also that a VISA session to a given device provides automatic offsets and bounds checking: If you require absolute addressing, use a session to the `VISA MEMACC` resource. Low-level register-based I/O operations such as `MapVXIAddress()` and `VXIpeek()` are also provided in VISA, with corresponding functions like `viMapAddress()` and `viPeekX()`. In NI-VISA, only one mapped region is allowed per session; open multiple sessions to a given device to map multiple regions for low-level access.

## Asserting and Handling Events

NI-VISA provides operations such as `viAssertIntrSignal()`, `viAssertTrigger()`, etc. to support the same functionality offered by `AssertVXIint()`, `SrcTrig()`, etc. in the NI-VXI API. These functions may use a VISA attribute to replace a feature provided by a parameter of the corresponding NI-VXI API operation.

The difference between the NI-VISA API and the NI-VXI API is most noticeable when programming to handle events such as VXI interrupts, signals, and triggers. The functions to enable and disable events are quite similar: NI-VISA provides `viEnableEvent()` and `viDisableEvent()` to correspond to the NI-VXI APIs proliferation of `Enable...()/Disable()` functions such as `EnableSignalInt()`, etc.

You can use both APIs to receive events asynchronously with a handler (also known as a callback) or to put the event into a queue. In NI-VISA, `viInstallHandler()` and the `VI_HNDLR` mechanism for `viEnableEvent()` indicate that a callback should be used, versus `VI_QUEUE` for queued events. This corresponds to the NI-VXI API list of `Set...Handler()` functions such as `SetVXIintHandler()`, etc., and the `RouteSignal()` function, which allows signals to be routed to a queue. Retrieving events from a queue is very similar: `viWaitOnEvent()` replaces `WaitForSignal()` and related functions.

You can also use both APIs to handle VXI interrupts and signals together, with NI-VISA providing the `VI_EVENT_VXI_SIGP` event type (which combines interrupts and signals) and NI-VXI providing the `RouteVXIint()` operation.

## Closing Your Application

At the end of the application, use `viClose()` to close the handle to device, and `viClose()` again on the default resource manager session to shut down NI-VISA, instead of `CloseVXIlibrary()`.

In summary, using NI-VISA instead of the NI-VXI API to control VXI devices will seem very natural for most applications. Although porting the code from NI-VXI to NI-VISA is not always completely straightforward, the easier and more powerful API and VISA's ability to provide compatibility across interfaces should be worth the change.



**Note** Some functionality, such as 48-bit Extended Word Serial protocol commands and SEMI-SYNC triggers, is available only in the NI-VXI API. If you require this functionality in your VISA program, contact National Instruments. We would like to hear more about your application and help you find a VISA-oriented solution.

## **Program a GPIB-VXI/C with NI-VISA**

You program a GPIB-VXI/C using the same function calls you would use to program any instrument in NI-VISA. You can perform message-based communication using `viWrite()` and `viRead()`. You can perform register-based communication using `viInX()`, `viOutX()`, `viMoveInX()`, and `viMoveOutX()` functions. See also [Communicate with message-based devices in NI-VISA or the NI-VXI API](#) and [Using NI-VISA](#).

## **Move a large block of data in NI-VISA or the NI-VXI API**

The most efficient way to move a block of data is to use `viMoveX()` in NI-VISA or `VXImove()` in the NI-VXI API. (National Instruments recommends using NI-VISA, rather than the NI-VXI API, to develop your application.) You can find an example of how to use these functions in the NI-VISA and NI-VXI API examples directories. The NI-VISA example is in the `vxipnp\win32\Nivisa\examples` directory and is called `HighReg.c`. The NI-VXI API example in the `nivxi\win32\examples` directory is called `VXIblock.c`.

## Optimizing Large VXIbus Transfers

For best performance, keep the following in mind when using `viMove()` or `VXImove()`:

- Make sure your buffers are 32-bit aligned.
- Transfer 32-bit data whenever possible.
- Use VXI block access privileges to significantly improve performance to devices that can accept block transfers, and likewise use D64 access privileges for devices that can accept the VME64 64-bit data transfer protocol.
- To optimize move performance on virtual memory systems such as the Windows operating system, lock the user buffer in memory yourself so the move operation does not need to lock the buffer.
- To optimize move performance on paged memory systems such as the Windows operating system, use a contiguous buffer so the move operation does not need to build a scatter-gather list for the user buffer.



**Note** `viMemAlloc()` or `VXImemAlloc()` returns 32-bit aligned, page-locked, continuous buffers that work efficiently with the move operations.

## Transfer data using VME64

NI-VISA supports VME64 by setting the appropriate attribute (`VI_ATTR_SRC_ACCESS_PRIV` or `VI_ATTR_DEST_ACCESS_PRIV`) using `viSetAttribute()`. Then you can call `viMoveX()` to transfer the data.



**Note** National Instruments recommends using NI-VISA, rather than the NI-VXI API, to develop your applications.

You can do a VME64 block transfer using the NI-VXI API function `VXImove()`. By setting bits 2 to 4 to 110 (for a nonprivileged access; for a privileged access, use 111), `VXImove()` issues VME64 block cycles to the specified address. Only certain devices can accept VME64 transfers. You should check the specifications of a particular instrument to see if it supports VME64.

See also [Move a large block of data in NI-VISA or the NI-VXI API](#).

## **Debug a VXI program**

Perhaps the best way to debug a VXI program is using the NI Spy utility. Launch NI Spy and configure it to log your function calls. Now when you run your program, NI Spy logs all your calls to NI-VISA, the NI-VXI API, or NI-488.2. It highlights any functions that return errors, and you can check the parameters that get passed into any of these functions.

If a device is not behaving the way you would expect it to, you can debug many problems using VISAIC or VIC. You can use these utilities to interact with your devices without writing a program. This way you can make sure your techniques for communicating with a device are sound without worrying about your application being written correctly. Once you can establish communication, you can use the same calls you made in VISAIC or VIC in your program.

## **Write a VXI program in C**

[Using instrument drivers](#)

[Using NI-VISA](#)

## Using instrument drivers (in C)

You can use *VXIplug&play* instrument drivers under Windows in the following supported C compilers: LabWindows/CVI, Microsoft Visual C++, and Borland C++.

The distribution disk of a WIN framework instrument driver contains the C source code files (.C and .H), a function panel file (.FP) that describes the graphical panels associated with the instrument driver, a DLL file (.DLL) which is a compiled version of the driver, and a Windows help file (.HLP).

To use the functions of the instrument driver, you must include the .H file in your C code. Unless you are using LabWindows/CVI, you need to link in the compiler-specific import library (.LIB) file to your project and copy the .DLL file to the Windows/System or WinNT/System directory.

Your program's first task is to initialize the connection to the instrument by calling the INITIALIZE function. One of the outputs of the INITIALIZE function is an Instrument ID, which is essentially a *handle* to the instrument. Once you have initialized your instrument connection, you can then either use the high-level APPLICATION functions, which perform complete measurement operations, or the lower-level COMPONENT functions, which give you more granularity in controlling the instrument's operation. At the end of your program, you should use the CLOSE function to close the connection to the instrument.

An important feature to take advantage of when using your *VXIplug&play* instrument driver is the online help included with the driver. The help describes each component of the driver and shows a hierarchical outline of its components.

## Using NI-VISA (in C)

NI-VISA supports the following C compilers under Windows: LabWindows/CVI, Microsoft Visual C++, and Borland C++. Reference the appropriate examples for each compiler. The examples are in the `vxipnp\win95\NIvisa\examples` or `vxipnp\winNT\NIvisa\examples\` directory. The basic flow of a NI-VISA program is the same under any compiler.

The first function you should call is `viOpenDefaultRM()`. This initializes NI-VISA and gives you a handle to the VISA Resource Manager that you need in later functions calls. To access an instrument, you obtain a handle to it using `viOpen()`. You then can set up any asynchronous event handlers that your program may need, then perform your accesses to all your instruments. When you are done, you should call `viClose()` on each instrument handle and the Resource Manager handle. Be sure to `#include visa.h` so that all your NI-VISA functions are declared.

Unless you are using LabWindows/CVI, you need to link in the compiler-specific import library `visa32.lib`. LabWindows/CVI takes care of this step.

See also [Communicate with message-based devices in NI-VISA or the NI-VXI API](#).

# Write a VXI program in LabVIEW

Using instrument drivers

Using NI-VISA

## Using instrument drivers (in LabVIEW)

LabVIEW can use both GWIN Framework instrument drivers and WIN Framework instrument drivers. If you have a GWIN driver for your instrument, you have access to block diagram source code from within LabVIEW. When you use a WIN driver in LabVIEW, the connection to the driver is accomplished via a DLL interface, so you do not have block diagram source code. But you do have native LabVIEW VIs, complete with front panels, icons, help information, and block diagrams (that call the DLL) for each function of the driver.

The first task your program should do is to initialize the connection to the instrument by calling the INITIALIZE function. One of the outputs of the INITIALIZE function is an Instrument ID, which is essentially a *handle* to the instrument. Once you have initialized your instrument connection, you can then use either the high-level APPLICATION Functions, which perform complete measurement operations, or the lower-level COMPONENT functions, which give you more granularity in controlling the instrument's operation. At the end of your program, you should use the CLOSE function to close the connection to the instrument.

To use a GWIN instrument driver in LabVIEW, you need the following files and software:

- LabVIEW version 5.0 or later for Windows
- VISA32.DLL version 2.01 or later
- instr\_name.LLB
- instr\_name.HLP

The instrument driver VIs can be loaded just as any other LabVIEW VIs.

To use a WIN instrument driver in LabVIEW, you need the following files and software:

- LabVIEW version 5.0 or later for Windows
- VISA32.DLL version 2.01 or later
- instr\_name.DLL
- instr\_name.H
- instr\_name.HLP

- instr\_name.LLB

WIN drivers are written in C using LabWindows/CVI. So, before you can use a WIN instrument driver in LabVIEW, you must use LabVIEW to convert the driver into LabVIEW VIs automatically. Launch LabVIEW and select **File»Convert CVI FP File...** (LabVIEW 5.x) or **Tools»Instrumentation»Import CVI Instrument Driver** (LabVIEW 6.x). You are prompted for both the .FP and .DLL files. Once you have converted a WIN instrument driver into a LabVIEW library (.LLB file), you can use the VIs like any other LabVIEW VIs.

Take advantage of the online help when using your *VXIplug&play* instrument driver. The help describes each component of the driver and shows a hierarchical outline of its components. When using LabVIEW, you can use the standard Windows Help system to access the instrument's Windows Help file.

## Using NI-VISA (in LabVIEW)

Examples of VXI programs written in LabVIEW using NI-VISA are in the `examples\instr\visa` directory.

The first VI you should call in your program is **VISA Open**. This VI opens a communication channel with an instrument. This communication channel is known as a session and is used throughout the rest of your VISA program to keep the unique communication and attribute settings of that instrument. You need to call the **VISA Open** VI for each instrument you want to communicate with. Once the sessions are open, you can set up any asynchronous event handlers that your program may need, and then perform your accesses to your instruments. When you are done, call the **VISA Close** VI on each session and event object.

You can use the **Simple Error Handler** VI to perform error checking and display errors when they occur.

**Write a VXI program in Visual Basic**

[Using NI-VISA](#)

## Using NI-VISA (in Visual Basic)

You must include the VISA32.BAS module in your Microsoft Visual Basic (version 4.0 or higher) application project file. This module is in the vxipnp\win95\include or vxipnp\winNT\include directory and contains the NI-VISA function prototypes for interfacing with the dynamic link library VISA32.DLL and VISA-specific constant definitions.

The VXI Visual Basic examples using VISA are in the ni-vxi\win32\vb\examples directory. You can use them as a reference to get started with your program.

The first function you should call is viOpenDefaultRM(). This initializes NI-VISA and gives you a handle to the VISA Resource Manager that you need in later function calls. After opening a session to the VISA Resource Manager, you can call viFindRsrc() to find out what instruments are available to open a session to. To access an instrument, open a session to it using viOpen(). Once the sessions to the instruments are open, you can set up any asynchronous event handlers that your program may need, and then perform your accesses to all your instruments. When you are done, call viClose() on each instrument handle and on the Resource Manager handle.

If you are writing an instrument driver, you should also include the module VPPTYPE.BAS in your application project file. This file is the *VXIplug&play* instrument driver header file and is in the vxipnp\win95\include or vxipnp\winNT\include directory.

## **Why Can't I**

[Configuration Questions](#)

[Programming Questions](#)

[Troubleshooting](#)

## **Configuration Questions**

[See my GPIB devices](#)

[See my Serial devices](#)

[See my VXI devices](#)

[Communicate with my GPIB-VXI/VXI devices?](#)

[Have the Resource Manager configure my VME devices](#)

[Share more than 8 MB of memory in A24 space](#)

[Use HP-VISA and NI-VISA on the same system](#)

## **See my GPIB devices**

There are several reasons why you may not be able to see your GPIB devices. First of all, you must have NI-488.2 correctly installed on your system. NI-VISA attempts to find all devices attached to the system. Some devices cannot be detected dynamically (particularly devices that do not respond to the find all listeners GPIB command).

## See my Serial devices

There are several reasons why you may not be able to see your serial devices. First of all, you must have [NI-VISA](#) correctly installed on your system. NI-VISA attempts to find all serial ports attached to the system. Ports are listed even if a device is not attached. Some ports cannot be detected dynamically (particularly parallel ports). To allow NI-VISA to see these devices, you must statically add them to the configuration. For serial ports on LabVIEW RT systems, you also need NI-Serial for LabVIEW RT installed.

## See my VXI devices

If you are using a GPIB-VXI to access your VXI chassis, see [How do I communicate with my GPIB-VXI/VXI devices](#).

If you are using an external or embedded VXI controller to access the VXI bus, you must have [NI-VXI](#) correctly installed on your system. The configuration utility uses the NI-VXI [Resman](#) utility to detect and configure your VXI devices. If Resman reports an error, your system may not be configured correctly.

If you have a device that should be detected, make sure that the chassis is turned on and that the instrument appears to be functioning. Check the MAX configuration view for warning or error messages. Resource conflicts between devices can cause Resman to fail. If you are using MXI-2 or the VXI-8340 series, make sure that the cable is securely attached on all ends, and (for MXI-2) that the correct end of the cable is plugged into your chassis. Verify this by checking that the label on the MXI-2 cable is closest to the MXI bus system controller.

Also, Resman cannot detect VME devices. If your system contains VME devices, you need to add these to the system manually.

## Have the Resource Manager configure my VME devices

The [Resource Manager](#) configures the system as defined by the VXI specification by reading and writing the VXI-defined [configuration registers](#). VME devices do not have these configuration registers, so Resource Manager does not know how to configure them.

To configure your system correctly to work with VME devices, you need to add them to the system manually. Your devices are then visible to NI-VISA, the NI-VXI API, and our utilities. This also prevents resource conflicts between VXI devices and your VME devices, allows the utility to notify you of potential conflicts, and correctly configures any VME-MXI-2 extenders.

## **Communicate with my GPIB-VXI/VXI devices?**

If you are using a GPIB-VXI to access your VXI chassis, you must have [NI-VISA](#) correctly installed on your system. NI-VISA cannot automatically find GPIB-VXI boards that are not set to GPIB primary address 1 and GPIB secondary address 0. If you have a GPIB-VXI board with GPIB settings other than these, you must manually add it to the system.

## **Share more than 8 MB of memory in A24 space**

The VXI specification allows any particular device to request no more than half of the address space in which its memory resides. Because A24 space is only 16 MB, the most memory any device can request from Resource Manager is 8 MB.

## **Use HP-VISA and NI-VISA on the same system**

HP-VISA and NI-VISA define the same API, but the implementation of each VISA function is different. Having both implementations of VISA on the same system would confuse your application. For example, when you call `viOpen` from LabVIEW, LabVIEW accesses whichever VISA driver that it finds.

## **Programming Questions**

Call MapVXIAddress() or viMapAddress()

Call VXImemAlloc() or viMemAlloc()

## Call MapVXIAddress() or viMapAddress()



**Note** National Instruments recommends using NI-VISA, rather than the NI-VXI API, to develop your applications.

MapVXIAddress or viMapAddress can fail if the size of your user window is set to zero or is less than the size of the window you are trying to map. You can check the size of your user window by right-clicking on your external controller or VXIpc embedded controller and selecting **Hardware Configuration**. The size of this window should exceed the sum of the sizes of the windows you are trying to map.

Check the sample code that illustrates how to use MapVXIAddress() and viMapAddress(). The NI-VISA sample code is in the vxipnp\win32\Nivisa\examples directory and is called LowReg.c. The NI-VXI API code, VXIlow.c, is in the nivxi\win32\msc(borlandc)\examples\ directory.

## Call VXImemAlloc() or viMemAlloc()



**Note** National Instruments recommends using NI-VISA, rather than the NI-VXI API, to develop your applications.

VXImemAlloc or viMemAlloc fail if it cannot allocate and lock down the amount of physical memory you request in the size parameter. If you are using shared memory, you can check these settings by right-clicking on your external controller or VXIpc embedded controller and selecting **Hardware Configuration**. The system on which you are sharing memory must not be set to A16 only. You can share system memory only in A24 or A32 space. The amount of memory you are sharing should be set large enough to handle all the memory you may want to allocate.

The amount of memory that you are sharing must be greater than the amount you are trying to allocate with VXImemAlloc() or viMemAlloc(). You must also reserve an amount of memory equal to or greater than the amount you are trying to allocate by setting the **Reserve physical memory** field in the **Shared Memory** page.

There is sample code that shows how to use VXImemAlloc and viMemAlloc(). The NI-VISA sample code is in the vxipnp\win32\Nivisa\examples directory and is called ShareSys.c. The NI-VXI API code, VXImem.c, is in the nivxi\win32\msc(borlandc)\examples\ directory.



**Note** Not all controllers support shared memory. For more information about shared memory support, refer to the documentation for your controller.

## Troubleshooting

[Device Busy](#)

[Device Disabled](#)

[Device Missing Functionality](#)

[Device Offline](#)

[Device OK](#)

[Device State Unknown](#)

[Device Static](#)

## **Device Busy**

This device is busy, and VISA cannot communicate with it. If this device is a COMM port, it may be in use with a modem, a mouse, or some other device. You may want to disable this device so that VISA does not try to open it in the future.

If you think this device should not be busy, try closing all applications or restarting Windows to kill any applications that may be using the device.

## Device Disabled

MAX could not determine the state of this device because it has been disabled. Disabled devices are not visible from VISA or your application. If the device is present in your system, and you want to use it, you can re-enable it by checking the **Device enabled** box in the properties dialog.

## **Device Missing Functionality**

This device may be missing some functionality. You may not experience any problems using this device if your application does not require the unavailable functionality.

## **Device Offline**

This device has failed its self-test and has been forced offline. You can try resetting the device and re-executing Resource Manager. If that does not help, contact the device vendor to find out more information.

## **Device OK**

This device is working properly. You should be able to communicate with it.

## **Device State Unknown**

Sometimes VISA cannot determine the state of a device. Devices that cannot be found dynamically or devices that do not implement a protocol for determining status would fall into this category.

## **Device Static**

MAX could not determine the state of this device because you statically added it to your system. VISA cannot dynamically detect static devices, and thus MAX cannot query for the device state.

## **Why Would I/When Do I**

Use NI-VISA vs. the NI-VXI API

Use NI-VISA vs. HP/Agilent VISA

Use low-level VXI access calls (use viMapAddress())

Use high-level VXI access calls

Use viMove() or VXImove()

Turn off ibIn in VISA

Not auto detect GPIB-VXI/C boards in VISA

Call viClose()

## **Use NI-VISA vs. the NI-VXI API**

National Instruments recommends using NI-VISA, rather than the NI-VXI API, to develop your VXI/VME applications. Both APIs can perform message-based communication with VXI instruments, do high- and low-level communication over the VXI backplane, perform block transfers, and handle asynchronous events. However, NI-VXI (that is, the software package that ships with National Instruments controllers and includes utilities such as Measurement & Automation Explorer and Resman) is optimized for use through NI-VISA.

If you are using GPIB or serial instruments, or if your VXI instruments are connected via a GPIB-VXI/C, you probably want to use NI-VISA because the API is interface independent. The calls you use to send messages to GPIB instruments are the same as those used to send messages to VXI instruments. NI-VISA also supports asynchronous I/O operations, whereas the NI-VXI API does not.

The NI-VXI API supports certain legacy instruments and behaviors that NI-VISA does not support, such as RORA interrupters and asynchronous notification of bus errors (note that VISA does report bus errors from high-level bus access functions).

You can use both interfaces in the same program, though it is not recommended unless you need functionality from both APIs. You should not try to handle asynchronous events with both languages. If you need to handle asynchronous events, such as VXI interrupts or triggers, you should choose one API and use that for all asynchronous operations.

## Use NI-VISA vs. HP/Agilent VISA

Whether you use [NI-VISA](#) or [HP/Agilent VISA](#) depends on the manufacturer of your GPIB and VXI controllers. The difference between NI-VISA and HP/Agilent VISA is in the interface to the hardware. If you are using National Instruments VXI and GPIB controllers, you should use NI-VISA. HP/Agilent VISA would not know how to communicate with a National Instruments board such as an external controller, for example. Conversely, if you are using an HP GPIB or VXI controller, you should use HP/Agilent VISA. You cannot use a GPIB controller board and a VXI controller from a different manufacturer together with VISA. The API for HP/Agilent VISA and NI-VISA is the same, however; thus, to the programmer there is no difference when writing code for one or the other.

You may use National Instruments and Hewlett-Packard GPIB-VXI controllers with either implementation of VISA. When using a GPIB-VXI in your system, the manufacturer of the GPIB controller board determines which VISA you should use.

## **Use low-level VXI access calls (use viMapAddress())**

Low-level VXI access functions provide the lowest level interface to the hardware and result in the best peek/poke performance, but are not as easy to use as high-level functions. `viMapAddress()` maps a window in your virtual address space to a portion of the VXI address space. You then have direct access to the memory on your backplane, but you are also responsible for managing the context of that window, checking for bus errors, and making sure you don't read/write beyond the boundaries of your window.

Many people use low-level calls when they shouldn't. If an area of your VXI address space is not time critical, do not map a window to it. Instead, communicate with those devices using the high-level calls. Also, many people place `viPeekX()/viPokeX()` calls in a loop to transfer blocks of data. This is usually not the most efficient method of doing block transfers. Instead, use the block transfer function `viMoveX()` to perform this task.

Should you choose to use low-level access calls in your program, you should look at the examples National Instruments provides.

## **Use high-level VXI access calls**

High-level VXI calls are the easiest way to establish register-level communication with VXI devices. You should use high-level calls whenever possible. If this is your first time programming in VXI, National Instruments recommends starting with high-level commands. If your application does speed-critical peeks and pokes, you may want to consider using the low-level access calls.

## Use viMove() or VXImove()

You should use viMoveX() (NI-VISA) or VXImove() (NI-VXI API) if you are transferring large blocks of data across the VXI backplane. (National Instruments recommends using NI-VISA, rather than the NI-VXI API, to develop your applications.) viMoveX() and VXImove() are the fastest ways to transfer a block of data because they take advantage of many features in the driver and in the hardware that are otherwise unavailable. Both of these functions can transfer data from contiguous blocks of memory and from FIFO queues.

See also [Move a large block of data in NI-VISA or the NI-VXI API](#).

## Turn off ibln in VISA

Certain NI-VISA calls (such as `viClear()` and `viAssertTrigger()`) perform an extra check using `ibln()` to confirm device presence. To further optimize the performance of your application, you can disable this feature in MAX.

From the **Tools** menu, select **NI-VISA»VISA Options** to bring up options you can configure in NI-VISA.

## **Not auto detect GPIB-VXI/C boards in VISA**

If you have a device at PA1 SA0 that is not a GPIB-VXI and does not respond to \*IDN?, disable auto detection.

From the **Tools** menu, select the **NI-VISA»VISA Options** to bring up options you can configure in NI-VISA.

## **Call viClose()**

This is the NI-VISA termination routine, which must be included at the end (or abort) of any application. You should call `viClose()` on any instrument handles that you have opened once you are done accessing that instrument. Calling `viClose()` on the Resource Manager handle disables interrupts and frees dynamic memory allocated for the internal Resource Manager table and other structures. Although you attempt to perform clean-up operations, failure to call `viClose()` on the Resource Manager handle when terminating your application can cause unpredictable and undesirable results. If your application can be aborted from some operating system abort routine, be certain to install an abort/close routine that calls `viClose()` on outstanding handles.

## **Important Information**

[Warranty](#)

[Copyright](#)

[Trademarks](#)

[Patents](#)

[Warning Regarding Use of NI Products](#)

## **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 THEREFORE 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.

National Instruments respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

## Trademarks

National Instruments, NI, ni.com, and LabVIEW are trademarks of National Instruments Corporation. Refer to the *Terms of Use* section on [ni.com/legal](http://ni.com/legal) for more information about [National Instruments trademarks](#).

FireWire® is the registered trademark of Apple Computer, Inc.

Handle Graphics®, MATLAB®, Real-Time Workshop®, Simulink®, Stateflow®, and xPC TargetBox® are registered trademarks, and TargetBox™ and Target Language Compiler™ are trademarks of The MathWorks, Inc.

Tektronix® and Tek are registered trademarks of Tektronix, Inc.

The Bluetooth® word mark is a registered trademark owned by the Bluetooth SIG, Inc.

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

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

## Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the patents.txt file on your media, or [ni.com/patents](http://ni.com/patents).

## **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.

## Technical Support and Professional Services

Visit the following sections of the award-winning National Instruments Web site at [ni.com](http://ni.com) for technical support and professional services:

- [Support](#)—Technical support resources at [ni.com/support](http://ni.com/support) include the following:
  - **Self-Help Resources**—For answers and solutions, visit [ni.com/support](http://ni.com/support) for software drivers and updates, a searchable [KnowledgeBase](#), [product manuals](#), step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on. Registered users also receive access to the [NI Discussion Forums](#) at [ni.com/forums](http://ni.com/forums). NI Applications Engineers make sure every question submitted online receives an answer.
  - **Standard Service Program Membership**—This program entitles members to direct access to NI Applications Engineers via phone and email for one-to-one technical support, as well as exclusive access to on demand training modules via the [Services Resource Center](#). NI offers complementary membership for a full year after purchase, after which you may renew to continue your benefits.  
  
For information about other [technical support options](#) in your area, visit [ni.com/services](http://ni.com/services) or [contact](#) your local office at [ni.com/contact](http://ni.com/contact).
- [Training and Certification](#)—Visit [ni.com/training](http://ni.com/training) for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- [System Integration](#)—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit [ni.com/alliance](http://ni.com/alliance).

If you searched [ni.com](http://ni.com) and could not find the answers you need, contact your [local office](#) or NI corporate headquarters. You also can visit the [Worldwide Offices](#) section of [ni.com/niglobal](http://ni.com/niglobal) to access the branch office

Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

## Branch Offices

Office	Telephone Number
Australia	1800 300 800
Austria	43 662 457990-0
Belgium	32 (0) 2 757 0020
Brazil	55 11 3262 3599
Canada	800 433 3488
China	86 21 5050 9800
Czech Republic	420 224 235 774
Denmark	45 45 76 26 00
Finland	358 (0) 9 725 72511
France	33 (0) 1 57 66 24 24
Germany	49 89 7413130
India	91 80 41190000
Israel	972 0 3 6393737
Italy	39 02 41309277
Japan	0120-527196 / 81 3 5472 2970
Korea	82 02 3451 3400
Lebanon	961 (0) 1 33 28 28
Malaysia	1800 887710
Mexico	01 800 010 0793
Netherlands	31 (0) 348 433 466
New Zealand	0800 553 322
Norway	47 (0) 66 90 76 60
Poland	48 22 3390150
Portugal	351 210 311 210
Russia	7 495 783 6851
Singapore	1800 226 5886
Slovenia	386 3 425 42 00
South Africa	27 0 11 805 8197
Spain	34 91 640 0085
Sweden	46 (0) 8 587 895 00
Switzerland	41 56 2005151
Taiwan	886 02 2377 2222
Thailand	662 278 6777
Turkey	90 212 279 3031
United Kingdom	44 (0) 1635 523545
United States (Corporate)	512 683 0100