VISA

NI-VISA™ Programmer Reference Manual

March 2000 Edition
Part Number 370132A-01

Worldwide Technical Support and Product Information

www.ni.com

National Instruments Corporate Headquarters

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

Worldwide Offices

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

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

Important Information

Warranty

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

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

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

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

Copyright

Trademarks

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

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

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

Contents

Chapter 4
Events

Chapter 5
Operations

About This Manual

This manual describes the attributes, events, and operations that comprise the VISA Application Programming Interface (API). This manual is meant to be used with the *NI-VISA User Manual*.

How To Use the Manual Set

Use the documentation that came with your GPIB and/or VXI hardware and software kit to install and configure your system.

Refer to the *Read Me First* document for information on installing the NI-VISA distribution media.

Use the *NI-VISA User Manual* for detailed information on how to program using VISA.

Use the *NI-VISA Programmer Reference Manual* for specific information about the attributes, events, and operations, such as format, syntax, parameters, and possible errors.

Conventions

The following conventions appear in this manual:

<> — Angle brackets that contain numbers separated by an ellipsis represent a range of values associated with a bit or signal name—for example, DBIO<3..0>.

[ ] — Square brackets enclose optional items—for example, [`response`].

» — 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.

♦ — The ♦ symbol indicates that the following text applies only to a specific product, a specific operating system, or a specific software version.

 — This icon denotes a note, which alerts you to important information.

bold	Bold text denotes items that you must select or click on in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.
`monospace bold`	Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.
`monospace italic`	Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

Related Documentation

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

- ANSI/IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation*
- ANSI/IEEE Standard 488.2-1992, *IEEE Standard Codes, Formats, Protocols, and Common Commands*
- ANSI/IEEE Standard 1014-1987, *IEEE Standard for a Versatile Backplane Bus: VMEbus*
- ANSI/IEEE Standard 1155-1992, *VMEbus Extensions for Instrumentation: VXIbus*
- ANSI/ISO Standard 9899-1990, *Programming Language C*
- *NI-488.2 Function Reference Manual for DOS/Windows*, National Instruments Corporation
- *NI-488.2 User Manual for Windows*, National Instruments Corporation
- *NI-VXI Programmer Reference Manual*, National Instruments Corporation
- *NI-VXI User Manual*, National Instruments Corporation

- *PXI Specification: PCI eXtensions for Instrumentation*, National Instruments Corporation
- VPP-1, *Charter Document*
- VPP-2, *System Frameworks Specification*
- VPP-3.1, *Instrument Drivers Architecture and Design Specification*
- VPP-3.2, *Instrument Driver Developers Specification*
- VPP-3.3, *Instrument Driver Function Panel Specification*
- VPP-4.3, *The VISA Library*
- VPP-4.3.2, *VISA Implementation Specification for Textual Languages*
- VPP-4.3.3, *VISA Implementation Specification for the G Language*
- VPP-6, *Installation and Packaging Specification*
- VPP-7, *Soft Front Panel Specification*
- VPP-8, *VXI Module/Mainframe to Receiver Interconnection*
- VPP-9, *Instrument Vendor Abbreviations*
- VXI-11, *TCP/IP Instrument Protocol*, *VXIbus Consortium*

1 Introduction

This chapter lists what you need to get started and presents a brief overview of VISA.

What You Need to Get Started

- ❑ Appropriate hardware support in the form of a National Instruments GPIB, GPIB-VXI, MXI/VXI or serial interface board. For serial support, the computer's standard serial ports are sufficient.
- ❑ NI-488.2 and/or NI-VXI installed on your system. For serial support, the system's serial drivers are sufficient.
- ❑ NI-VISA distribution media.
- ❑ If you have a GPIB-VXI command module from another vendor, you need that vendor's GPIB-VXI VISA component.
- ❑ To download the latest version of the NI-VISA distribution media, point your web browser to `www.ni.com/visa`.

VXI*plug&play* Overview

TheVXI*plug&play* Systems Alliance was formed on September 22, 1993 with the goal of increasing end-user success and multivendor interoperability for VXIbus systems. To achieve this goal, VXI*plug&play* defines and implements new levels of standardization to simplify multivendor VXI system integration to benefit both end-users and vendors. As a result, VXI*plug&play* products are easy to use, thanks to new standards for both hardware and software.

At the heart of these standards is the Virtual Instrument Software Architecture, or *VISA*, the I/O software standard on which all VXI*plug&play* software components are based. In the past, there were many different I/O software products to control GPIB and VXI. Software written with these various libraries supplied by individual vendors was directly and uniquely tied to the hardware these vendors produced. The

VISA standard, endorsed by over 35 of the largest instrumentation companies in the industry including Tektronix, Agilent, and National Instruments, unifies the industry to make software interoperable, reusable, and able to stand the test of time.

When the VISA standard was initially endorsed, commercial VISA products were not yet available. To quickly realize the benefits of VXI*plug&play*, the alliance developed the VISA Transition Library (VTL) specification. The VTL reflected the alliance's strategy to deliver multi-vendor software interoperability, while at the same time moving the entire industry towards a common, robust VISA foundation for the future. Software written to VTL, such as instrument drivers and executable soft front panels, will also run on present and future VISA implementations without modification.

Supported Platforms

This manual and the *NI-VISA User Manual* describe how to use NI-VISA, the National Instruments implementation of the VISA I/O standard, in any environment using LabWindows/CVI, any ANSI C compiler, or Microsoft Visual Basic. NI-VISA currently supports the frameworks and programming languages shown in Table 1-1. For information on programming VISA from LabVIEW, refer to the VISA documentation included with your LabVIEW distribution.

Table 1-1. NI-VISA Support

Operating System	Programming Language/ Environment	Framework
Windows 98/95	LabWindows/CVI, ANSI C, Visual Basic	WIN95
	LabVIEW	GWIN95
Windows NT 4.0/ Windows 2000	LabWindows/CVI, ANSI C, Visual Basic	WINNT
	LabVIEW	GWINNT
Solaris 2.*x*, Solaris 7.*x* (32-bit)	LabWindows/CVI, ANSI C	SUN
	LabVIEW	GSUN
HP-UX 10.*x*, HP-UX 11.*x* (32-bit)	ANSI C, LabWindows/CVI*	HPUX
	LabVIEW	GHPUX

Table 1-1. NI-VISA Support (Continued)

Operating System	Programming Language/ Environment	Framework
Mac PPC	ANSI C	**
	LabVIEW	**
Linux *x*86	ANSI C	**
	LabVIEW	**
VxWorks (Intel)	ANSI C	**

* Although the LabWindows/CVI development environment is not available on HP-UX, the run-time libraries are. Therefore, a LabWindows/CVI application developed on another framework can be ported to HP-UX without modification.

** This framework is not defined by the VXI*plug&play* Systems Alliance, but is still supported by NI-VISA.

The VXI*plug&play* Systems Alliance developed the concept of a *framework* to categorize operating systems, programming languages, and I/O software libraries to bring the most useful products to the most end-users. A framework is a logical grouping of the choices that you face when designing a VXI system. You must always choose an operating system and a programming language along with an application development environment (ADE) when building a system. There are trade-offs associated with each of these decisions; many configurations are possible. The VXI*plug&play* Systems Alliance grouped the most popular operating systems, programming languages, and ADEs into distinct frameworks and defined in-depth specifications to guarantee interoperability of the components within each framework. To claim VXI*plug&play* compliance, a component must be compliant within a given framework.

With this version of NI-VISA, you can perform message-based and register-based communication with instruments, assert triggers, share memory, and respond to interrupts and triggers. For VXI, you can also perform register accesses at the interface level and mainframe-specific control and monitoring of utility lines. For GPIB, you can also perform board-level commands and the control and monitoring of bus lines. NI-VISA provides all the I/O functionality that you need for your test and measurement application.

2

Overview of the VISA API

This chapter contains an overview of the VISA Application Programming Interface (API).

You can use this manual as a reference to the VISA API. This API is partitioned into three distinct mechanisms that access information on a given resource: attributes, events, and operations.

VISA Access Mechanisms

The following paragraphs summarize the most important characteristics of attributes, events, and operations. Please refer to Chapter 3, *VISA Overview,* in the *NI-VISA User Manual* for a more detailed description of this subject.

Attributes

An attribute describes a value within a session or resource that reflects a characteristic of the operational state of the given object. These attributes are accessed through the following operations:

- `viGetAttribute()`
- `viSetAttribute()`

Events

An event is an asynchronous occurrence that is independent of the normal sequential execution of the process running in a system. Depending on how you want to handle event occurrences, you can use the `viEnableEvent()` operation with either the `viInstallHandler()` operation or the `viWaitOnEvent()` operation.

Events respond to attributes in the same manner that resources do. Once your application is done using a particular event received via `viWaitOnEvent()`, it should call `viClose()` to destroy that event.

Operations

An operation is an action defined by a resource that can be performed on the given resource. Each resource has the ability to define a series of operations. In addition to those defined by each resource you can use the following template operations in any resource:

- `viClose()`
- `viGetAttribute()`
- `viSetAttribute()`
- `viStatusDesc()`
- `viTerminate()`
- `viLock()`
- `viUnlock()`
- `viEnableEvent()`
- `viDisableEvent()`
- `viDiscardEvents()`
- `viWaitOnEvent()`
- `viInstallHandler()`
- `viUninstallHandler()`

VISA Resource Types

Currently, there are several VISA resource types—INSTR, MEMACC, INTFC, BACKPLANE, SERVANT, and SOCKET. Most VISA applications and instrument drivers use only the INSTR Resource.

INSTR

A VISA Instrument Control (INSTR) Resource lets a controller interact with the device associated with the given resource. This resource type grants the controller the following services to perform message-based and/or register-based I/O, depending on the type of device and the interface to which the device is connected.

Basic I/O services include the ability to send and receive blocks of data to and from the device. The meaning of the data is device dependent, and could be a message, command, or other binary encoded data. For devices compliant with IEEE-488, the basic I/O services also include triggering (both software and hardware), servicing requests, reading status bytes, and clearing the device.

Formatted I/O services provide both formatted and buffered I/O capabilities for data transfers to and from devices. The formatting capabilities include those specified by ANSI C, with extensions for common protocols used by instrumentation systems. Buffering improves system performance by making it possible to not only transfer large blocks of data, but also send several commands at one time.

Memory I/O (or Register I/O) services allow register-level access to devices connected to interfaces that support direct memory access, such as the VXIbus or VMEbus. Both high-level and low-level access services have operations for individual register accesses, with a trade-off between speed and complexity. The high-level access services also have operations for moving large blocks of data to and from devices. When using an INSTR Resource, all address parameters are relative to the device's assigned memory base in the given address space; knowing a device's base address is neither required by nor relevant to the user.

Shared Memory services make it possible to allocate memory on a particular device that is to be used exclusively by a given session. This is usually available only on devices that export shared memory specifically for such a purpose, such as a VXIbus or VMEbus controller.

MEMACC

A VISA Memory Access (MEMACC) Resource lets a controller interact with the interface associated with the given resource. Advanced users who need to perform memory accesses directly between multiple devices typically use the MEMACC Resource. This resource type gives the controller the following services to access arbitrary registers or memory addresses on memory-mapped buses.

Memory I/O (or Register I/O) services allow register level access to interfaces that support direct memory access, such as the VXIbus or VMEbus. Both high-level and low-level access services have operations for individual register accesses, with a trade-off between speed and complexity. The high-level access services also have operations for moving large blocks of data to and from arbitrary addresses. When using a MEMACC Resource, all address parameters are absolute within the given address space; knowing a device's base address is both required by and relevant to the user.

INTFC

A VISA GPIB Bus Interface (INTFC) Resource lets a controller interact with any devices connected to the board associated with the given resource. Advanced GPIB users who need to control multiple devices simultaneously or need to have multiple controllers in a single system typically use the INTFC Resource. This resource type provides basic and formatted I/O services as described below. In addition, the controller can directly query and manipulate specific lines on the bus, and also pass control to other devices with controller capability.

Basic I/O services include the ability to send and receive blocks of data onto and from the bus. The meaning of the data is device dependent, and could be a message, command, or other binary encoded data. The basic I/O services also include triggering devices on the bus and sending miscellaneous commands to any or all devices.

Formatted I/O services provide both formatted and buffered I/O capabilities for data transfers to and from devices. The formatting capabilities include those specified by ANSI C, with extensions for common protocols used by instrumentation systems. Buffering improves system performance by making it possible to not only transfer large blocks of data, but also send several commands at one time.

BACKPLANE

A VISA VXI Mainframe Backplane (BACKPLANE) Resource encapsulates the operations and properties of each mainframe (or chassis) in a VXIbus system. This resource type lets a controller query and manipulate specific lines on a specific mainframe in a given VXI system. BACKPLANE services allow the user to map, unmap, assert, and receive hardware triggers, and also to assert and receive various utility and interrupt signals. This includes advanced functionality that might not be available in all implementations or on all controllers.

SERVANT

A VISA Servant (SERVANT) Resource encapsulates the operations and properties of the capabilities of a device and a device's view of the system in which it exists. The SERVANT Resource exposes the device-side functionality of the device associated with the given resource. The SERVANT Resource is a class for advanced users who want to write firmware code that exports message-based device functionality across potentially multiple interfaces. This resource type provides basic and formatted I/O services as described below.

Basic I/O services include the ability to receive blocks of data from a commander and respond with blocks of data in return. The meaning of the data is device dependent, and could be a message, command, or other binary encoded data. The basic I/O services also include setting a 488-style status byte and receiving device clear and trigger events.

Formatted I/O services provide both formatted and buffered I/O capabilities for data transfers from and to the given device's commander. The formatting capabilities include those specified by ANSI C, with extensions for common protocols used by instrumentation systems. Buffering improves system performance by making it possible to not only transfer large blocks of data, but also send several commands at one time.

A VXI Servant Resource also provides services to assert and receive various utility and interrupt signals.

SOCKET

A VISA Ethernet Socket (SOCKET) Resource encapsulates the operations and properties of the capabilities of a raw Ethernet connection using TCP/IP. The SOCKET Resource exposes the capability of a raw socket connection over TCP/IP. This resource type provides basic and formatted I/O services as described below.

Basic I/O services include the ability to send and receive blocks of data to and from the device. The meaning of the data is device dependent, and could be a message, command, or other binary encoded data. If the device is capable of communicating with 488.2-style strings, the basic I/O services also include software triggering, querying a 488-style status byte, and sending a device clear message.

Formatted I/O services provide both formatted and buffered I/O capabilities for data transfers to and from devices. The formatting capabilities include those specified by ANSI C, with extensions for common protocols used by instrumentation systems. Buffering improves system performance by making it possible to not only transfer large blocks of data, but also send several commands at one time.

Description of the API

The following three chapters describe the individual attributes, events, and operations. These are listed in alphabetical order within each access mechanism. Since a particular item can refer to more than one resource or interface type, each item is clearly marked with the resource and interface that support it.

Refer to Appendix B, *Resources*, for a quick reference of how the attributes, events, and operations map to the available resources.

3

Attributes

This chapter describes the VISA attributes. The attribute descriptions are listed in alphabetical order for easy reference.

Each attribute description contains a list below the title indicating the supported resource classes, such as GPIB, Serial, etc. The Attribute Information table lists the access privilege, the data type, range of values, and the default value.

VI_ATTR_ASRL_AVAIL_NUM

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViUInt32`	0 to FFFFFFFFh	N/A

Description

`VI_ATTR_ASRL_AVAIL_NUM` shows the number of bytes available in the global receive buffer.

Related Items

See the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_ASRL_BAUD

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViUInt32	0 to FFFFFFFFh	9600

Description

VI_ATTR_ASRL_BAUD is the baud rate of the interface. It is represented as an unsigned 32-bit integer so that any baud rate can be used, but it usually requires a commonly used rate such as 300, 1200, 2400, or 9600 baud.

Related Items

See the VI_ATTR_ASRL_DATA_BITS, VI_ATTR_ASRL_FLOW_CNTRL, VI_ATTR_ASRL_PARITY, and VI_ATTR_ASRL_STOP_BITS descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_ASRL_CTS_STATE

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED(1) VI_STATE_UNASSERTED(0) VI_STATE_UNKNOWN(-1)	N/A

Description

VI_ATTR_ASRL_CTS_STATE shows the current state of the Clear To Send (CTS) input signal.

Related Items

See the VI_ATTR_ASRL_DCD_STATE, VI_ATTR_ASRL_DSR_STATE, VI_ATTR_ASRL_DTR_STATE, VI_ATTR_ASRL_RI_STATE, and VI_ATTR_ASRL_RTS_STATE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_ASRL_DATA_BITS

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViUInt16	5 to 8	8

Description

VI_ATTR_ASRL_DATA_BITS is the number of data bits contained in each frame (from 5 to 8). The data bits for each frame are located in the low-order bits of every byte stored in memory.

Related Items

See the VI_ATTR_ASRL_BAUD, VI_ATTR_ASRL_FLOW_CNTRL, VI_ATTR_ASRL_PARITY, and VI_ATTR_ASRL_STOP_BITS descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_ASRL_DCD_STATE

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED(1) VI_STATE_UNASSERTED(0) VI_STATE_UNKNOWN(-1)	N/A

Description

VI_ATTR_ASRL_DCD_STATE shows the current state of the Data Carrier Detect (DCD) input signal. The DCD signal is often used by modems to indicate the detection of a carrier (remote modem) on the telephone line. The DCD signal is also known as *Receive Line Signal Detect* (RLSD).

Related Items

See the VI_ATTR_ASRL_CTS_STATE, VI_ATTR_ASRL_DSR_STATE, VI_ATTR_ASRL_DTR_STATE, VI_ATTR_ASRL_RI_STATE, and VI_ATTR_ASRL_RTS_STATE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_ASRL_DSR_STATE

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED(1) VI_STATE_UNASSERTED(0) VI_STATE_UNKNOWN(-1)	N/A

Description

VI_ATTR_ASRL_DSR_STATE shows the current state of the Data Set Ready (DSR) input signal.

Related Items

See the VI_ATTR_ASRL_CTS_STATE, VI_ATTR_ASRL_DCD_STATE, VI_ATTR_ASRL_DTR_STATE, VI_ATTR_ASRL_RI_STATE, and VI_ATTR_ASRL_RTS_STATE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_ASRL_DTR_STATE

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViInt16	VI_STATE_ASSERTED(1) VI_STATE_UNASSERTED(0) VI_STATE_UNKNOWN(-1)	N/A

Description

VI_ATTR_ASRL_DTR_STATE shows the current state of the Data Terminal Ready (DTR) input signal. When the VI_ATTR_ASRL_FLOW_CNTRL attribute is set to VI_ASRL_FLOW_DTR_DSR, this attribute is ignored when changed, but can be read to determine whether the background flow control is asserting or unasserting the signal.

Related Items

See the VI_ATTR_ASRL_CTS_STATE, VI_ATTR_ASRL_DCD_STATE, VI_ATTR_ASRL_DSR_STATE, VI_ATTR_ASRL_RI_STATE, and VI_ATTR_ASRL_RTS_STATE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_ASRL_END_IN

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_ASRL_END_NONE(0) VI_ASRL_END_LAST_BIT(1) VI_ASRL_END_TERMCHAR(2)	VI_ASRL_END_TERMCHAR

Description

VI_ATTR_ASRL_END_IN indicates the method used to terminate read operations.

- If it is set to VI_ASRL_END_NONE, the read will not terminate until all of the requested data is received (or an error occurs).
- If it is set to VI_ASRL_END_LAST_BIT, the read will terminate as soon as a character arrives with its last bit set. For example, if VI_ATTR_ASRL_DATA_BITS is set to 8, the read will terminate when a character arrives with the 8th bit set.
- If it is set to VI_ASRL_END_TERMCHAR, the read will terminate as soon as the character in VI_ATTR_TERMCHAR is received. In this case, VI_ATTR_TERMCHAR_EN is ignored.

Because the default value of VI_ATTR_TERMCHAR is 0Ah (linefeed), read operations on serial ports will stop reading whenever a linefeed is encountered. To change this behavior, you must change the value of one of these attributes—VI_ATTR_ASRL_END_IN or VI_ATTR_TERMCHAR.

Related Items

See the VI_ATTR_ASRL_END_OUT and VI_ATTR_TERMCHAR descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_ASRL_END_OUT

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_ASRL_END_NONE(0) VI_ASRL_END_LAST_BIT(1) VI_ASRL_END_TERMCHAR(2) VI_ASRL_END_BREAK(3)	VI_ASRL_END_NONE

Description

VI_ATTR_ASRL_END_OUT indicates the method used to terminate write operations.

- If it is set to VI_ASRL_END_NONE, the write will not append anything to the data being written.
- If it is set to VI_ASRL_END_LAST_BIT, the write will send all but the last character with the last bit clear, then transmit the last character with the last bit set. For example, if VI_ATTR_ASRL_DATA_BITS is set to 8, the write will clear the 8th bit for all but the last character, then transmit the last character with the 8th bit set.
- If it is set to VI_ASRL_END_TERMCHAR, the write will send the character in VI_ATTR_TERMCHAR after the data being transmitted.
- If it is set to VI_ASRL_END_BREAK, the write will transmit a break after all the characters for the write have been sent.

Related Items

See the VI_ATTR_ASRL_END_IN and VI_ATTR_TERMCHAR descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_ASRL_FLOW_CNTRL

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViUInt16	VI_ASRL_FLOW_NONE(0) VI_ASRL_FLOW_XON_XOFF(1) VI_ASRL_FLOW_RTS_CTS(2) VI_ASRL_FLOW_DTR_DSR(4)	VI_ASRL_FLOW_NONE

Description

VI_ATTR_ASRL_FLOW_CNTRL indicates the type of flow control used by the transfer mechanism.

- If this attribute is set to VI_ASRL_FLOW_NONE, the transfer mechanism does not use flow control, and buffers on both sides of the connection are assumed to be large enough to hold all data transferred.
- If this attribute is set to VI_ASRL_FLOW_XON_XOFF, the transfer mechanism uses the XON and XOFF characters to perform flow control. The transfer mechanism controls input flow by sending XOFF when the receive buffer is nearly full, and it controls the output flow by suspending transmission when XOFF is received.
- If this attribute is set to VI_ASRL_FLOW_RTS_CTS, the transfer mechanism uses the RTS output signal and the CTS input signal to perform flow control. The transfer mechanism controls input flow by unasserting the RTS signal when the receive buffer is nearly full, and it controls output flow by suspending the transmission when the CTS signal is unasserted.
- If this attribute is set to VI_ASRL_FLOW_DTR_DSR, the transfer mechanism uses the DTR output signal and the DSR input signal to perform flow control. The transfer mechanism controls input flow by unasserting the DTR signal when the receive buffer is nearly full, and it controls output flow by suspending the transmission when the DSR signal is unasserted.

This attribute can specify multiple flow control mechanisms by bit-ORing multiple values together. However, certain combinations may not be supported by all serial ports and/or operating systems.

Related Items

See the VI_ATTR_ASRL_BAUD, VI_ATTR_ASRL_DATA_BITS, VI_ATTR_ASRL_PARITY, VI_ATTR_ASRL_STOP_BITS, VI_ATTR_ASRL_XON_CHAR, and VI_ATTR_ASRL_XOFF_CHAR, descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_ASRL_PARITY

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViUInt16	VI_ASRL_PAR_NONE(0) VI_ASRL_PAR_ODD(1) VI_ASRL_PAR_EVEN(2) VI_ASRL_PAR_MARK(3) VI_ASRL_PAR_SPACE(4)	VI_ASRL_PAR_NONE

Description

VI_ATTR_ASRL_PARITY is the parity used with every frame transmitted and received.

- VI_ASRL_PAR_MARK means that the parity bit exists and is always 1.
- VI_ASRL_PAR_SPACE means that the parity bit exists and is always 0.

Related Items

See the VI_ATTR_ASRL_BAUD, VI_ATTR_ASRL_DATA_BITS, VI_ATTR_ASRL_FLOW_CNTRL, and VI_ATTR_ASRL_STOP_BITS descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_ASRL_REPLACE_CHAR

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt8	0 to FFh	0

Description

VI_ATTR_ASRL_REPLACE_CHAR specifies the character to be used to replace incoming characters that arrive with errors (such as parity error).

Related Items

See the VI_ATTR_ASRL_PARITY description in this chapter. See the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_ASRL_RI_STATE

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED(1) VI_STATE_UNASSERTED(0) VI_STATE_UNKNOWN(–1)	N/A

Description

VI_ATTR_ASRL_RI_STATE shows the current state of the Ring Indicator (RI) input signal. The RI signal is often used by modems to indicate that the telephone line is ringing.

Related Items

See the VI_ATTR_ASRL_CTS_STATE, VI_ATTR_ASRL_DCD_STATE, VI_ATTR_ASRL_DSR_STATE, VI_ATTR_ASRL_DTR_STATE, and VI_ATTR_ASRL_RTS_STATE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_ASRL_RTS_STATE

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViInt16	VI_STATE_ASSERTED (1) VI_STATE_UNASSERTED (0) VI_STATE_UNKNOWN (–1)	N/A

Description

VI_ATTR_ASRL_RTS_STATE is used to manually assert or unassert the Request To Send (RTS) output signal. When the VI_ATTR_ASRL_FLOW_CNTRL attribute is set to VI_ASRL_FLOW_RTS_CTS, this attribute is ignored when changed, but can be read to determine whether the background flow control is asserting or unasserting the signal.

Related Items

See the VI_ATTR_ASRL_CTS_STATE, VI_ATTR_ASRL_DCD_STATE, VI_ATTR_ASRL_DSR_STATE, VI_ATTR_ASRL_DTR_STATE, and VI_ATTR_ASRL_RI_STATE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_ASRL_STOP_BITS

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViUInt16	VI_ASRL_STOP_ONE(10) VI_ASRL_STOP_ONE5(15) VI_ASRL_STOP_TWO(20)	VI_ASRL_STOP_ONE

Description

VI_ATTR_ASRL_STOP_BITS is the number of stop bits used to indicate the end of a frame. The value VI_ASRL_STOP_ONE5 indicates one-and-one-half (1.5) stop bits.

Related Items

See the VI_ATTR_ASRL_BAUD, VI_ATTR_ASRL_DATA_BITS, VI_ATTR_ASRL_FLOW_CNTRL, and VI_ATTR_ASRL_PARITY descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_ASRL_XOFF_CHAR

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt8	0 to FFh	<Control-S> (13h)

Description

VI_ATTR_ASRL_XOFF_CHAR specifies the value of the XOFF character used for XON/XOFF flow control (both directions). If XON/XOFF flow control (software handshaking) is not being used, the value of this attribute is ignored.

Related Items

See the VI_ATTR_ASRL_XON_CHAR and VI_ATTR_ASRL_FLOW_CNTRL descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_ASRL_XON_CHAR

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt8	0 to FFh	<Control-Q> (11h)

Description

VI_ATTR_ASRL_XON_CHAR specifies the value of the XON character used for XON/XOFF flow control (both directions). If XON/XOFF flow control (software handshaking) is not being used, the value of this attribute is ignored.

Related Items

See the VI_ATTR_ASRL_XOFF_CHAR and VI_ATTR_ASRL_FLOW_CNTRL descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_BUFFER

Resource Classes

VI_EVENT_IO_COMPLETION

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViBuf	N/A	N/A

Description

VI_ATTR_BUFFER contains the address of a buffer that was used in an asynchronous operation.

Related Items

See the VI_ATTR_STATUS, VI_ATTR_JOB_ID, and VI_ATTR_RET_COUNT descriptions in this chapter. See the VI_EVENT_IO_COMPLETION event description in Chapter 4, *Events*.

VI_ATTR_CMDR_LA

Resource Classes

GPIB-VXI INSTR, VXI INSTR, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	0 to 255 VI_UNKNOWN_LA (–1)	N/A

Description

VI_ATTR_CMDR_LA is the unique logical address of the commander of the VXI device used by the given session.

Related Items

See the *INSTR Resource* and *SERVANT Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_DEST_ACCESS_PRIV

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, VXI INSTR, VXI MEMACC

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_DATA_PRIV(0) VI_DATA_NPRIV(1) VI_PROG_PRIV(2) VI_PROG_NPRIV(3) VI_BLCK_PRIV(4) VI_BLCK_NPRIV(5) VI_D64_PRIV(6) VI_D64_NPRIV(7)	VI_DATA_PRIV

Description

VI_ATTR_DEST_ACCESS_PRIV specifies the address modifier to be used in high-level access operations, such as viOut*XX*() and viMoveOut*XX*(), when writing to the destination.

Related Items

See the VI_ATTR_DEST_BYTE_ORDER, VI_ATTR_DEST_INCREMENT, VI_ATTR_SRC_ACCESS_PRIV, and VI_ATTR_WIN_ACCESS_PRIV descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_DEST_BYTE_ORDER

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, VXI INSTR, VXI MEMACC

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_BIG_ENDIAN(0) VI_LITTLE_ENDIAN(1)	VI_BIG_ENDIAN

Description

VI_ATTR_DEST_BYTE_ORDER specifies the byte order to be used in high-level access operations, such as viOut*XX*() and viMoveOut*XX*(), when writing to the destination.

Related Items

See the VI_ATTR_DEST_ACCESS_PRIV, VI_ATTR_DEST_INCREMENT, VI_ATTR_SRC_BYTE_ORDER, and VI_ATTR_WIN_BYTE_ORDER descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_DEST_INCREMENT

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, PXI INSTR, VXI INSTR, VXI MEMACC

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	`ViInt32`	0 to 1	1

Description

`VI_ATTR_DEST_INCREMENT` is used in the `viMoveOut`*`XX`*`()` operations to specify by how many elements the destination offset is to be incremented after every transfer. The default value of this attribute is 1 (that is, the destination address will be incremented by 1 after each transfer), and the `viMoveOut`*`XX`*`()` operations move into consecutive elements. If this attribute is set to 0, the `viMoveOut`*`XX`*`()` operations will always write to the same element, essentially treating the destination as a FIFO register.

Related Items

See the `VI_ATTR_DEST_ACCESS_PRIV`, `VI_ATTR_DEST_BYTE_ORDER`, and `VI_ATTR_SRC_INCREMENT` descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_DEV_STATUS_BYTE

Resource Classes

GPIB INTFC, GPIB SERVANT, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	`ViUInt8`	0 to FFh	N/A

Description

This attribute specifies the 488-style status byte of the local controller or device associated with this session.

If this attribute is written and bit 6 (40h) is set, this device or controller will assert a service request (SRQ) if it is defined for this interface.

Related Items

See *INTFC Resource* and *SERVANT Resource* in Appendix B, *Resources*.

VI_ATTR_DMA_ALLOW_EN

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, PXI INSTR, Serial INSTR, TCPIP INSTR, VXI INSTR, VXI MEMACC, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	`ViBoolean`	`VI_TRUE (1)` `VI_FALSE (0)`	N/A

Description

This attribute specifies whether I/O accesses should use DMA (VI_TRUE) or Programmed I/O (VI_FALSE). In some implementations, this attribute may have global effects even though it is documented to be a local attribute. Since this affects performance and not functionality, that behavior is acceptable.

Related Items

See *MEMACC Resource*, *INTFC Resource*, *SERVANT Resource*, and *INSTR Resource* in Appendix B, *Resources*.

VI_ATTR_EVENT_TYPE

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	`ViEventType`	0h to FFFFFFFFh	N/A

Description

`VI_ATTR_EVENT_TYPE` is the unique logical identifier for the event type of the specified event.

Related Items

Refer to Chapter 4, *Events*, for a list of events.

VI_ATTR_FDC_CHNL

Resource Classes

VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	0 to 7	N/A

Description

VI_ATTR_FDC_CHNL determines which Fast Data Channel (FDC) will be used to transfer the buffer.

Related Items

See the VI_ATTR_FDC_MODE and VI_ATTR_FDC_USE_PAIR descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_FDC_MODE

Resource Classes

VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_FDC_NORMAL(1) VI_FDC_STREAM(2)	VI_FDC_NORMAL

Description

VI_ATTR_FDC_MODE specifies which Fast Data Channel (FDC) mode to use (either normal or stream mode).

Related Items

See the VI_ATTR_FDC_CHNL and VI_ATTR_FDC_USE_PAIR descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_FDC_USE_PAIR

Resource Classes

VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE(1) VI_FALSE(0)	VI_FALSE

Description

Setting VI_ATTR_FDC_USE_PAIR to VI_TRUE specifies to use a channel pair for transferring data. Otherwise, only one channel will be used.

Related Items

See the VI_ATTR_FDC_CHNL and VI_ATTR_FDC_MODE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_FILE_APPEND_EN

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	`ViBoolean`	`VI_TRUE(1)` `VI_FALSE(0)`	`VI_FALSE`

Description

This attribute specifies whether `viReadToFile()` will overwrite (truncate) or append when opening a file.

Related Items

See `viReadToFile()` in Chapter 5, *Operations*. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* in Appendix B, *Resources*.

VI_ATTR_GPIB_ADDR_STATE

Resource Classes

GPIB INTFC, GPIB SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_GPIB_UNADDRESSED(0) VI_GPIB_TALKER(1) VI_GPIB_LISTENER(2)	N/A

Description

This attribute shows whether the specified GPIB interface is currently addressed to talk or listen, or is not addressed.

Related Items

Also see the *INTFC Resource* and *SERVANT Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_GPIB_ATN_STATE

Resource Classes

GPIB INTFC

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED(1) VI_STATE_UNASSERTED(0) VI_STATE_UNKNOWN(–1)	N/A

Description

This attribute shows the current state of the GPIB ATN (ATtentioN) interface line.

Related Items

See VI_ATTR_GPIB_REN_STATE, VI_ATTR_GPIB_NDAC_STATE and VI_ATTR_GPIB_SRQ_STATE in this chapter. Also see the *INTFC Resource* description in Appendix B, *Resources*.

VI_ATTR_GPIB_CIC_STATE

Resource Classes

GPIB INTFC

Attribute Information

Access Privilege	Data Type	Range	Default
Read-only Global	ViBoolean	VI_TRUE(1) VI_FALSE(0)	N/A

Description

This attribute shows whether the specified GPIB interface is currently CIC (Controller In Charge).

Related Items

See VI_ATTR_GPIB_SYS_CNTRL_STATE in this chapter. Also see the *INTFC Resource* description in Appendix B, *Resources*.

VI_ATTR_GPIB_HS488_CBL_LEN

Resource Classes

GPIB INTFC

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViInt16	VI_GPIB_HS488_NIMPL(-1) VI_GPIB_HS488_DISABLED(0) 1-15	N/A

Description

This attribute specifies the total number of meters of GPIB cable used in the specified GPIB interface.

Related Items

See VI_ATTR_IO_PROT in this chapter. Also see *INTFC Resource* in Appendix B, *Resources*.

VI_ATTR_GPIB_NDAC_STATE

Resource Classes

GPIB INTFC

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED(1) VI_STATE_UNASSERTED(0) VI_STATE_UNKNOWN(–1)	N/A

Description

This attribute shows the current state of the GPIB NDAC (Not Data Accepted) interface line.

Related Items

See VI_ATTR_GPIB_REN_STATE, VI_ATTR_GPIB_ATN_STATE and VI_ATTR_GPIB_SRQ_STATE in this chapter. Also see the *INTFC Resource* description in Appendix B, *Resources*.

VI_ATTR_GPIB_PRIMARY_ADDR

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE

Attribute Information

Access Privilege	Data Type	Range	Default
INSTR, MEMACC, BACKPLANE: Read Only Global **INTFC, SERVANT:** Read/Write Global	`ViUInt16`	0 to 30	N/A

Description

`VI_ATTR_GPIB_PRIMARY_ADDR` specifies the primary address of the GPIB device used by the given session. For the GPIB INTFC and GPIB SERVANT Resources, this attribute is Read-Write.

Related Items

See the `VI_ATTR_GPIB_SECONDARY_ADDR` description in this chapter. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, *BACKPLANE Resource*, and *MEMACC Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_GPIB_READDR_EN

Resource Classes

GPIB INSTR, GPIB-VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE(1) VI_FALSE(0)	VI_TRUE

Description

VI_ATTR_GPIB_READDR_EN specifies whether to use repeat addressing before each read or write operation.

Related Items

See the VI_ATTR_GPIB_UNADDR_EN description in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_GPIB_RECV_CIC_STATE

Resource Classes

VI_EVENT_GPIB_CIC

Attribute Information

Access Privilege	Data Type	Range	Default
Read-Only	ViBoolean	VI_TRUE(1) VI_FALSE(0)	N/A

Description

This attribute specifies whether the local controller has gained or lost CIC status.

Related Items

See the VI_ATTR_GPIB_ATN_STATE description in this chapter. See VI_EVENT_GPIB_CIC in Chapter 4, *Events*. Also see the *INTFC Resource* description in Appendix B, *Resources*.

VI_ATTR_GPIB_REN_STATE

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViBoolean	VI_TRUE(1) VI_FALSE(0)	N/A

Description

VI_ATTR_GPIB_REN_STATE returns the current state of the GPIB REN (Remote ENable) interface line.

Related Items

See VI_ATTR_GPIB_ATN_STATE, VI_ATTR_GPIB_NDAC_STATE and VI_ATTR_GPIB_SRQ_STATE in this chapter. Also see the *INSTR Resource*, *INTFC Resource* and *SERVANT Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_GPIB_SECONDARY_ADDR

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE

Attribute Information

Access Privilege	Data Type	Range	Default
INSTR, MEMACC, BACKPLANE: Read Only Global **INTFC, SERVANT:** Read/Write Global	ViUInt16	0 to 30, VI_NO_SEC_ADDR (FFFFh)	N/A

Description

VI_ATTR_GPIB_SECONDARY_ADDR specifies the secondary address of the GPIB device used by the given session. For the GPIB INTFC and GPIB SERVANT Resources, this attribute is Read-Write.

Related Items

See the VI_ATTR_GPIB_PRIMARY_ADDR description in this chapter. Also see the *INSTR Resource*, *INTFC Resource*, *MEMACC Resource*, *BACKPLANE Resource*, and *SERVANT Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_GPIB_SRQ_STATE

Resource Classes

GPIB INTFC

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED(1) VI_STATE_UNASSERTED(0) VI_STATE_UNKNOWN(–1)	N/A

Description

This attribute shows the current state of the GPIB SRQ (Service ReQuest) interface line.

Related Items

See VI_ATTR_GPIB_REN_STATE, VI_ATTR_GPIB_NDAC_STATE, and VI_ATTR_GPIB_ATN_STATE in this chapter. Also see the *INTFC Resource* description in Appendix B, *Resources*.

VI_ATTR_GPIB_SYS_CNTRL_STATE

Resource Classes

GPIB INTFC

Attribute Information

Access Privilege	Data Type	Range	Default
Read-Only Global	ViBoolean	VI_TRUE(1) VI_FALSE(0)	N/A

Description

This attribute shows whether the specified GPIB interface is currently the system controller. In some implementations, this attribute may be modified only through a configuration utility. On these systems this attribute is read-only (RO).

Related Items

See VI_ATTR_GPIB_CIC_STATE in this chapter. Also see the *INTFC Resource* in Appendix B, *Resources*.

VI_ATTR_GPIB_UNADDR_EN

Resource Classes

GPIB INSTR, GPIB-VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE(1) VI_FALSE(0)	VI_FALSE

Description

VI_ATTR_GPIB_UNADDR_EN specifies whether to unaddress the device (UNT and UNL) after each read or write operation.

Related Items

See the VI_ATTR_GPIB_READDR_EN description in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_IMMEDIATE_SERV

Resource Classes

GPIB-VXI INSTR, VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViBoolean	VI_TRUE(1) VI_FALSE(0)	N/A

Description

VI_ATTR_IMMEDIATE_SERV specifies whether the device associated with this session is an immediate servant of the controller running VISA.

Related Items

See the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_INTF_INST_NAME

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViString`	N/A	N/A

Description

`VI_ATTR_INTF_INST_NAME` specifies human-readable text that describes the given interface.

Related Items

See the `VI_ATTR_INTF_NUM` and `VI_ATTR_INTF_TYPE` descriptions in this chapter. Also see the *INSTR Resource*, *MEMACC Resource*, *INTFC Resource*, *BACKPLANE Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_INTF_NUM

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViUInt16`	0h to FFFFh	0

Description

`VI_ATTR_INTF_NUM` specifies the board number for the given interface.

Related Items

See the `VI_ATTR_INTF_TYPE` description in this chapter. Also see the *INSTR Resource*, *MEMACC Resource*, *INTFC Resource*, *BACKPLANE Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_INTF_PARENT_NUM

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0h to FFFFh	0

Description

VI_ATTR_INTF_PARENT_NUM specifies the board number of the GPIB board to which the GPIB-VXI is attached.

Related Items

See the VI_ATTR_INTF_NUM and VI_ATTR_INTF_TYPE descriptions in this chapter. Also see the *INTFC Resource*, *MEMACC Resource* and *BACKPLANE Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_INTF_TYPE

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	VI_INTF_GPIB(1) VI_INTF_VXI(2) VI_INTF_GPIB_VXI(3) VI_INTF_ASRL(4) VI_INTF_PXI(5) VI_INTF_TCPIP(6)	N/A

Description

VI_ATTR_INTF_TYPE specifies the interface type of the given session.

Related Items

See the VI_ATTR_INTF_NUM description in this chapter. Also see the *INSTR Resource*, *MEMACC Resource*, *INTFC Resource*, *BACKPLANE Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_INTR_STATUS_ID

Resource Classes

VI_EVENT_VXI_VME_INTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt32	0 to FFFFFFFFh	N/A

Description

VI_ATTR_INTR_STATUS_ID specifies the 32-bit status/ID retrieved during the IACK cycle.

Related Items

See the VI_ATTR_EVENT_TYPE and VI_ATTR_RECV_INTR_LEVEL descriptions in this chapter. See the VI_EVENT_VXI_VME_INTR event description in Chapter 4, *Events*. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_IO_PROT

Resource Classes

GPIB INTFC, GPIB INSTR, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	**GPIB**: VI_NORMAL(1) VI_HS488(3)	VI_NORMAL
		VXI: VI_NORMAL(1) VI_FDC(2)	VI_NORMAL
		GPIB-VXI: VI_NORMAL(1)	VI_NORMAL
		Serial, TCPIP: VI_NORMAL(1) VI_PROT_4882_STRS(4)	VI_NORMAL

Description

VI_ATTR_IO_PROT specifies which protocol to use. In VXI systems, for example, you can choose between normal word serial or Fast Data Channel (FDC). In GPIB, you can choose between normal and high-speed (HS488) data transfers. For a session to a Serial device or Ethernet socket, you can choose between normal and 488-style transfers. In previous versions of VISA, VI_PROT_4882_STRS was known as VI_ASRL488.

Related Items

See the VI_ATTR_FDC_CHNL, VI_ATTR_FDC_MODE, VI_ATTR_GPIB_HS488_CBL_LEN, and VI_ATTR_FDC_USE_PAIR descriptions in this chapter. Also see the *INSTR Resource*, *SOCKET Resource*, *SERVANT Resource*, and *INTFC Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_JOB_ID

Resource Classes

VI_EVENT_IO_COMPLETION

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	`ViJobId`	N/A	N/A

Description

`VI_ATTR_JOB_ID` contains the job ID of the asynchronous operation that has completed.

Related Items

See the `VI_ATTR_STATUS`, `VI_ATTR_BUFFER`, and `VI_ATTR_RET_COUNT` descriptions in this chapter. See the `VI_EVENT_IO_COMPLETION` event description in Chapter 4, *Events*. Also see Appendix B, *Resources*.

VI_ATTR_MAINFRAME_LA

Resource Classes

GPIB-VXI INSTR, GPIB-VXI BACKPLANE, VXI INSTR, VXI BACKPLANE

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	0 to 255 VI_UNKNOWN_LA(-1)	N/A

Description

VI_ATTR_MAINFRAME_LA specifies the lowest logical address in the mainframe. If the logical address is not known, VI_UNKNOWN_LA is returned.

Related Items

See the *INSTR Resource* and *BACKPLANE Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_MANF_ID

Resource Classes

GPIB-VXI INSTR, PXI INSTR, VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0h to FFFh	N/A

Description

VI_ATTR_MANF_ID is the manufacturer identification number of the device.

Related Items

See the VI_ATTR_MODEL_CODE, VI_ATTR_MANF_NAME, and VI_ATTR_PXI_SUB_MANF_ID descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_MANF_NAME

Resource Classes

GPIB-VXI INSTR, PXI INSTR, VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViString`	N/A	N/A

Description

This string attribute is the manufacturer's name. The value of this attribute should be used for display purposes only and not for programmatic decisions, as the value can be different between VISA implementations and/or revisions.

Related Items

See the `VI_ATTR_MANF_ID` and `VI_ATTR_MODEL_NAME` descriptions in this chapter. Also see the *INSTR Resource* description from Appendix B, *Resources*.

VI_ATTR_MAX_QUEUE_LENGTH

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	`ViUInt32`	1h to FFFFFFFFh	50

Description

`VI_ATTR_MAX_QUEUE_LENGTH` specifies the maximum number of events that can be queued at any time on the given session. Events that occur after the queue has become full will be discarded.

`VI_ATTR_MAX_QUEUE_LENGTH` is a Read/Write attribute until the first time `viEnableEvent()` is called on a session. Thereafter, this attribute is Read Only.

Related Items

See the `viEnableEvent()` and `viWaitOnEvent()` descriptions in Chapter 5, *Operations*. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

VI_ATTR_MEM_BASE

Resource Classes

GPIB-VXI INSTR, VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViBusAddress	0h to FFFFFFFFh	N/A

Description

VI_ATTR_MEM_BASE specifies the base address of the device in VXIbus memory address space. This base address is applicable to A24 or A32 address space. If the value of VI_ATTR_MEM_SPACE is VI_A16_SPACE, the value of this attribute is meaningless for the given VXI device.

Related Items

See the VI_ATTR_MEM_SIZE and VI_ATTR_MEM_SPACE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_MEM_SIZE

Resource Classes

GPIB-VXI INSTR, VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViBusSize	0h to FFFFFFFFh	N/A

Description

VI_ATTR_MEM_SIZE specifies the size of memory requested by the device in VXIbus address space. If the value of VI_ATTR_MEM_SPACE is VI_A16_SPACE, the value of this attribute is meaningless for the given VXI device.

Related Items

See the VI_ATTR_MEM_BASE and VI_ATTR_MEM_SPACE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_MEM_SPACE

Resource Classes

GPIB-VXI INSTR, VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	VI_A16_SPACE(1) VI_A24_SPACE(2) VI_A32_SPACE(3)	VI_A16_SPACE

Description

VI_ATTR_MEM_SPACE specifies the VXIbus address space used by the device. The three types are A16, A24, or A32 memory address space.

A VXI device with memory in A24 or A32 space also has registers accessible in the configuration section of A16 space. A VME device with memory in multiple address spaces requires one VISA resource for each address space used.

Related Items

See the VI_ATTR_MEM_BASE and VI_ATTR_MEM_SIZE descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_MODEL_CODE

Resource Classes

GPIB-VXI INSTR, PXI INSTR, VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0h to FFFFh	N/A

Description

VI_ATTR_MODEL_CODE specifies the model code for the device.

Related Items

See the VI_ATTR_PXI_SUB_MODEL_CODE, VI_ATTR_MANF_ID, and VI_ATTR_MODEL_NAME descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_MODEL_NAME

Resource Classes

GPIB-VXI INSTR, PXI INSTR, VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViString`	N/A	N/A

Description

This string attribute is the model name of the device. The value of this attribute should be used for display purposes only and not for programmatic decisions, as the value can be different between VISA implementations and/or revisions.

Related Items

See the `VI_ATTR_MODEL_CODE`, `VI_ATTR_PXI_SUB_MODEL_CODE`, and `VI_ATTR_MANF_NAME` descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_OPER_NAME

Resource Classes

VI_EVENT_IO_COMPLETION, VI_EVENT_EXCEPTION

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	`ViString`	N/A	N/A

Description

`VI_ATTR_OPER_NAME` contains the name of the operation generating this event.

Related Items

See the `VI_ATTR_STATUS` and `VI_ATTR_EVENT_TYPE` descriptions in this chapter. See the `VI_EVENT_EXCEPTION` and `VI_EVENT_IO_COMPLETION` event descriptions in Chapter 4, *Events*. Also see Appendix B, *Resources*.

VI_ATTR_PXI_DEV_NUM

Resource Classes

PXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViUInt16`	0 to 31	N/A

Description

This is the PXI device number.

Related Items

See `VI_ATTR_PXI_FUNC_NUM` in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_PXI_FUNC_NUM

Resource Classes

PXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0 to 7	0

Description

This is the PXI function number. All devices have a function 0. Multifunction devices will also support other function numbers.

Related Items

See VI_ATTR_PXI_DEV_NUM in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_PXI_MEM_BASE_BAR0/VI_ATTR_PXI_MEM_BASE_BAR1/ VI_ATTR_PXI_MEM_BASE_BAR2/VI_ATTR_PXI_MEM_BASE_BAR3/ VI_ATTR_PXI_MEM_BASE_BAR4/VI_ATTR_PXI_MEM_BASE_BAR5

Resource Classes

PXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt32	0 to FFFFFFFFh	N/A

Description

PXI memory base address assigned to the specified BAR. If the value of the corresponding VI_ATTR_PXI_MEM_TYPE_BAR*x* is VI_PXI_ADDR_NONE, the value of this attribute is meaningless for the given PXI device.

Related Items

See VI_ATTR_PXI_MEM_TYPE_BAR0/VI_ATTR_PXI_MEM_TYPE_BAR1/ VI_ATTR_PXI_MEM_TYPE_BAR2/VI_ATTR_PXI_MEM_TYPE_BAR3/ VI_ATTR_PXI_MEM_TYPE_BAR4/VI_ATTR_PXI_MEM_TYPE_BAR5 and VI_ATTR_PXI_MEM_SIZE_BAR0/VI_ATTR_PXI_MEM_SIZE_BAR1/ VI_ATTR_PXI_MEM_SIZE_BAR2/VI_ATTR_PXI_MEM_SIZE_BAR3/ VI_ATTR_PXI_MEM_SIZE_BAR4/VI_ATTR_PXI_MEM_SIZE_BAR5 in this chapter. Also see the *INSTR Resource* in Appendix B, *Resources*.

VI_ATTR_PXI_MEM_SIZE_BAR0/VI_ATTR_PXI_MEM_SIZE_BAR1/ VI_ATTR_PXI_MEM_SIZE_BAR2/VI_ATTR_PXI_MEM_SIZE_BAR3/ VI_ATTR_PXI_MEM_SIZE_BAR4/VI_ATTR_PXI_MEM_SIZE_BAR5

Resource Classes

PXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt32	0 to FFFFFFFFh	N/A

Description

Memory size used by the device in the specified BAR. If the value of the corresponding VI_ATTR_PXI_MEM_TYPE_BAR*x* is VI_PXI_ADDR_NONE, the value of this attribute is meaningless for the given PXI device.

Related Items

See VI_ATTR_PXI_MEM_TYPE_BAR0/VI_ATTR_PXI_MEM_TYPE_BAR1/ VI_ATTR_PXI_MEM_TYPE_BAR2/VI_ATTR_PXI_MEM_TYPE_BAR3/ VI_ATTR_PXI_MEM_TYPE_BAR4/VI_ATTR_PXI_MEM_TYPE_BAR5 and VI_ATTR_PXI_MEM_BASE_BAR0/VI_ATTR_PXI_MEM_BASE_BAR1/ VI_ATTR_PXI_MEM_BASE_BAR2/VI_ATTR_PXI_MEM_BASE_BAR3/ VI_ATTR_PXI_MEM_BASE_BAR4/VI_ATTR_PXI_MEM_BASE_BAR5 in this chapter. Also see the *INSTR Resource* in Appendix B, *Resources*.

VI_ATTR_PXI_MEM_TYPE_BAR0/VI_ATTR_PXI_MEM_TYPE_BAR1/ VI_ATTR_PXI_MEM_TYPE_BAR2/VI_ATTR_PXI_MEM_TYPE_BAR3/ VI_ATTR_PXI_MEM_TYPE_BAR4/VI_ATTR_PXI_MEM_TYPE_BAR5

Resource Classes

PXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	VI_PXI_ADDR_NONE(0) VI_PXI_ADDR_MEM(1) VI_PXI_ADDR_IO(2)	N/A

Description

Memory type used by the device in the specified BAR (if applicable).

Related Items

See VI_ATTR_PXI_MEM_SIZE_BAR0/VI_ATTR_PXI_MEM_SIZE_BAR1/ VI_ATTR_PXI_MEM_SIZE_BAR2/VI_ATTR_PXI_MEM_SIZE_BAR3/ VI_ATTR_PXI_MEM_SIZE_BAR4/VI_ATTR_PXI_MEM_SIZE_BAR5 and VI_ATTR_PXI_MEM_BASE_BAR0/VI_ATTR_PXI_MEM_BASE_BAR1/ VI_ATTR_PXI_MEM_BASE_BAR2/VI_ATTR_PXI_MEM_BASE_BAR3/ VI_ATTR_PXI_MEM_BASE_BAR4/VI_ATTR_PXI_MEM_BASE_BAR5 in this chapter. Also see the *INSTR Resource* in Appendix B, *Resources*.

VI_ATTR_PXI_SUB_MANF_ID

Resource Classes

PXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0 to FFFFh	N/A

Description

This attribute specifies the PXI device's subsystem manufacturer ID (if applicable).

Related Items

See VI_ATTR_MANF_ID, VI_ATTR_MANF_NAME, VI_ATTR_MODEL_CODE, VI_ATTR_PXI_SUB_MODEL_CODE in this chapter. Also see the *INSTR Resource* in Appendix B, *Resources*.

VI_ATTR_PXI_SUB_MODEL_CODE

Resource Classes

PXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0 to FFFFh	N/A

Description

This attribute specifies the PXI device's subsystem model code (if applicable).

Related Items

See VI_ATTR_MODEL_NAME, VI_ATTR_MODEL_CODE and VI_ATTR_PXI_SUB_MANF_ID in this chapter. Also see the *INSTR Resource* in Appendix B, *Resources*.

VI_ATTR_RD_BUF_OPER_MODE

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIC SOCKET, VXI INSTR, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_FLUSH_ON_ACCESS(1) VI_FLUSH_DISABLE(3)	VI_FLUSH_DISABLE

Description

VI_ATTR_RD_BUF_OPER_MODE specifies the operational mode of the formatted I/O read buffer. When the operational mode is set to VI_FLUSH_DISABLE (default), the buffer is flushed only on explicit calls to viFlush(). If the operational mode is set to VI_FLUSH_ON_ACCESS, the write buffer is flushed every time a viScanf() (or related) operation completes.

Related Items

See the VI_ATTR_WR_BUF_OPER_MODE description in this chapter. See the viFlush() and viScanf() descriptions in Chapter 5, *Operations*. Also see the *INSTR Resource*, *INTFC Resource*, *SOCKET Resource* and *SERVANT Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_RECV_INTR_LEVEL

Resource Classes

VI_EVENT_VXI_VME_INTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViInt16	1 to 7; VI_UNKNOWN_LEVEL(-1)	N/A

Description

VI_ATTR_RECV_INTR_LEVEL is the VXI interrupt level on which the interrupt was received.

Related Items

See the VI_ATTR_EVENT_TYPE and VI_ATTR_INTR_STATUS_ID descriptions in this chapter. See the VI_EVENT_VXI_VME_INTR event description in Chapter 4, *Events*. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_RECV_TRIG_ID

Resource Classes

VI_EVENT_TRIG

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViInt16	VI_TRIG_SW(-1) VI_TRIG_TTL0(0) to VI_TRIG_TTL7(7); VI_TRIG_ECL0(8) to VI_TRIG_ECL1(9)	N/A

Description

VI_ATTR_RECV_TRIG_ID identifies the triggering mechanism on which the specified trigger event was received.

Related Items

See the VI_EVENT_TRIG event description in Chapter 4, *Events*. Also see the *BACKPLANE Resource*, *INSTR Resource*, *INTFC Resource* and *SERVANT Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_RET_COUNT

Resource Classes

VI_EVENT_IO_COMPLETION

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViUInt32	0h to FFFFFFFFh	N/A

Description

VI_ATTR_RET_COUNT contains the actual number of elements that were asynchronously transferred.

Related Items

See the VI_ATTR_STATUS, VI_ATTR_JOB_ID, and VI_ATTR_BUFFER descriptions in this chapter. See the VI_EVENT_IO_COMPLETION event description in Chapter 4, *Events*. Also see Appendix B, *Resources*.

VI_ATTR_RM_SESSION

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Local	`ViSession`	N/A	N/A

Description

`VI_ATTR_RM_SESSION` specifies the session of the Resource Manager that was used to open this session.

Related Items

See the *VISA Resource Template* description in Appendix B, *Resources*.

VI_ATTR_RSRC_CLASS

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViString`	N/A	N/A

Description

`VI_ATTR_RSRC_CLASS` specifies the resource class (for example, "INSTR") as defined by the canonical resource name.

Related Items

See the `VI_ATTR_RSRC_NAME` description in this chapter. See the *VISA Resource Template* description in Appendix B, *Resources*.

VI_ATTR_RSRC_IMPL_VERSION

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViVersion`	0h to FFFFFFFFh	N/A

Description

`VI_ATTR_RSRC_IMPL_VERSION` is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

Related Items

See the `VI_ATTR_RSRC_SPEC_VERSION` description in this chapter. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

VI_ATTR_RSRC_LOCK_STATE

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViAccessMode`	`VI_NO_LOCK(0)` `VI_EXCLUSIVE_LOCK(1)` `VI_SHARED_LOCK(2)`	`VI_NO_LOCK`

Description

`VI_ATTR_RSRC_LOCK_STATE` indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

Related Items

See the *VISA Resource Template* description in Appendix B, *Resources*.

VI_ATTR_RSRC_MANF_ID

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViUInt16`	0h to 3FFFh	N/A

Description

`VI_ATTR_RSRC_MANF_ID` is a value that corresponds to the VXI manufacturer ID of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Related Items

See the `VI_ATTR_RSRC_MANF_NAME` description in this chapter. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

VI_ATTR_RSRC_MANF_NAME

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViString`	N/A	N/A

Description

`VI_ATTR_RSRC_MANF_NAME` is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

Related Items

See the `VI_ATTR_RSRC_MANF_ID` description in this chapter. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

VI_ATTR_RSRC_NAME

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViRsrc	N/A	N/A

Description

VI_ATTR_RSRC_NAME is the unique identifier for a resource compliant with the address structure shown in the following table. Optional string segments are shown in square brackets.

Interface	Syntax
GPIB INSTR	GPIB[*board*]::*primary address*[::*secondary address*][::INSTR]
GPIB INTFC	GPIB[*board*]::INTFC
GPIB SERVANT	GPIB[*board*]::SERVANT
GPIB-VXI INSTR	GPIB-VXI[*board*]::*VXI logical address*[::INSTR]
GPIB-VXI BACKPLANE	GPIB-VXI[*board*][::*mainframe logical address*]::BACKPLANE
GPIB-VXI MEMACC	GPIB-VXI[*board*]::MEMACC
PXI INSTR	PXI[*board*]::*device*[::*function*][::INSTR]
Serial INSTR	ASRL[*board*][::INSTR]
TCPIP INSTR	TCPIP [board]::*host address*[::LAN device name][::INSTR]
TCPIP SOCKET	TCPIP [board]::*host address*::*port*::SOCKET
VXI INSTR	VXI[*board*]::*VXI logical address*[::INSTR]
VXI BACKPLANE	VXI[*board*][::*mainframe logical address*]::BACKPLANE

Interface	Syntax
VXI MEMACC	VXI[*board*]::MEMACC
VXI SERVANT	VXI[*board*]::SERVANT

The following table shows examples of address strings as defined in the previous table.

Address String	Description
VXI0::1::INSTR	A VXI device at logical address 1 in VXI interface VXI0.
GPIB-VXI::9::INSTR	A VXI device at logical address 9 in a GPIB-VXI controlled system.
GPIB::1::0::INSTR	A GPIB device at primary address 1 and secondary address 0 in GPIB interface 0.
ASRL1::INSTR	A serial device located on port 1.
VXI::MEMACC	Board-level register access to the VXI interface.
GPIB-VXI1::MEMACC	Board-level register access to GPIB-VXI interface number 1.
GPIB2::INTFC	Interface or raw resource for GPIB interface 2.
VXI::1::BACKPLANE	Mainframe resource for chassis 1 on the default VXI system, which is interface 0.
GPIB-VXI2::BACKPLANE	Mainframe resource for default chassis on GPIB-VXI interface 2.
GPIB1::SERVANT	Servant/device-side resource for GPIB interface 1.
VXI0::SERVANT	Servant/device-side resource for VXI interface 0.
PXI::15::INSTR	PXI device number 15 on bus 0.
TCPIP0::1.2.3.4::999 ::SOCKET	Raw TCP/IP access to port 999 at the specified IP address.
TCPIP::dev@company.com ::INSTR	A TCP/IP device using VXI-11 located at the specified address. This uses the default LAN Device Name of inst0.

Related Items

See the viFindRsrc(), viParseRsrc(), and viOpen() description in Chapter 5, *Operations*. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

VI_ATTR_RSRC_SPEC_VERSION

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViVersion`	0h to FFFFFFFFh	00200200h

Description

`VI_ATTR_RSRC_SPEC_VERSION` is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00200200h.

Related Items

See the `VI_ATTR_RSRC_IMPL_VERSION` description in this chapter. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

VI_ATTR_SEND_END_EN

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE(1) VI_FALSE(0)	VI_TRUE

Description

VI_ATTR_SEND_END_EN specifies whether to assert END during the transfer of the last byte of the buffer.

Related Items

See the viWrite() description in Chapter 5, *Operations*, and the *INSTR Resource*, *INTFC Resource*, *SOCKET Resource* and *SERVANT Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_SIGP_STATUS_ID

Resource Classes

VI_EVENT_VXI_SIGP

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	`ViUInt16`	0h to FFFFh	N/A

Description

`VI_ATTR_SIGP_STATUS_ID` is the 16-bit Status/ID value retrieved during the IACK cycle or from the Signal register.

Related Items

See the `VI_EVENT_VXI_SIGP` event description in Chapter 4, *Events*. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_SLOT

Resource Classes

GPIB-VXI INSTR, PXI INSTR, VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	0 to 12 VI_UNKNOWN_SLOT(-1)	N/A

Description

VI_ATTR_SLOT specifies the physical slot location of the VXIbus device. If the slot number is not known, VI_UNKNOWN_SLOT is returned.

Related Items

See the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_SRC_ACCESS_PRIV

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, VXI INSTR, VXI MEMACC

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_DATA_PRIV(0) VI_DATA_NPRIV(1) VI_PROG_PRIV(2) VI_PROG_NPRIV(3) VI_BLCK_PRIV(4) VI_BLCK_NPRIV(5) VI_D64_PRIV(6) VI_D64_NPRIV(7)	VI_DATA_PRIV

Description

VI_ATTR_SRC_ACCESS_PRIV specifies the address modifier to be used in high-level access operations, such as viIn*XX*( ) and viMoveIn*XX*( ), when reading from the source.

Related Items

See the VI_ATTR_DEST_ACCESS_PRIV, VI_ATTR_SRC_BYTE_ORDER, VI_ATTR_SRC_INCREMENT, and VI_ATTR_WIN_ACCESS_PRIV descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_SRC_BYTE_ORDER

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, VXI INSTR, VXI MEMACC

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_BIG_ENDIAN(0) VI_LITTLE_ENDIAN(1)	VI_BIG_ENDIAN

Description

VI_ATTR_SRC_BYTE_ORDER specifies the byte order to be used in high-level access operations, such as viIn*XX*() and viMoveIn*XX*(), when reading from the source.

Related Items

See the VI_ATTR_DEST_BYTE_ORDER, VI_ATTR_SRC_ACCESS_PRIV, VI_ATTR_SRC_INCREMENT, and VI_ATTR_WIN_BYTE_ORDER descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_SRC_INCREMENT

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, PXI INSTR, VXI INSTR, VXI MEMACC

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	`ViInt32`	0 to 1	1

Description

`VI_ATTR_SRC_INCREMENT` is used in the `viMoveIn`*`XX`*`()` operations to specify by how many elements the source offset is to be incremented after every transfer. The default value of this attribute is 1 (that is, the source address will be incremented by 1 after each transfer), and the `viMoveOut`*`XX`*`()` operations move from consecutive elements. If this attribute is set to 0, the `viMoveIn`*`XX`*`()` operations will always read from the same element, essentially treating the source as a FIFO register.

Related Items

See the `VI_ATTR_DEST_INCREMENT`, `VI_ATTR_SRC_ACCESS_PRIV`, and `VI_ATTR_SRC_BYTE_ORDER` descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_STATUS

Resource Classes

VI_EVENT_EXCEPTION, VI_EVENT_IO_COMPLETION

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViStatus	N/A	N/A

Description

VI_ATTR_STATUS contains the return code of the operation generating this event.

Related Items

See the VI_ATTR_BUFFER, VI_ATTR_JOB_ID, VI_ATTR_OPER_NAME, and VI_ATTR_RET_COUNT descriptions in this chapter. See the VI_EVENT_EXCEPTION and VI_EVENT_IO_COMPLETION event descriptions in Chapter 4, *Events*. Also see Appendix B, *Resources*.

VI_ATTR_SUPPRESS_END_EN

Resource Classes

Serial INSTR, VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE(1) VI_FALSE(0)	VI_FALSE

Description

VI_ATTR_SUPPRESS_END_EN specifies whether to suppress the END bit termination. If this attribute is set to VI_TRUE, the END bit does not terminate read operations. If this attribute is set to VI_FALSE, the END bit terminates read operations.

Related Items

See the viRead( ) description in Chapter 5, *Operations*, and the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_TCPIP_ADDR

Resource Classes

TCPIP INSTR, TCPIP SOCKET

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViString`	N/A	N/A

Description

This is the TCPIP address of the device to which the session is connected. This string is formatted in dot notation.

Related Items

See the `VI_ATTR_TCPIP_HOSTNAME` description in this chapter. Also see the *INSTR Resource* and *SOCKET Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_TCPIP_DEVICE_NAME

Resource Classes

TCPIP INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViString`	N/A	N/A

Description

This specifies the LAN device name used by the VXI-11 protocol during connection.

Related Items

See the *INSTR Resource* description in Appendix B, *Resources*.

VI_ATTR_TCPIP_HOSTNAME

Resource Classes

TCPIP INSTR, TCPIP SOCKET

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViString`	N/A	N/A

Description

This specifies the host name of the device. If no host name is available, this attribute returns an empty string.

Related Items

See the *INSTR Resource* and *SOCKET Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_TCPIP_KEEPALIVE

Resource Classes

TCPIP SOCKET

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE(1) VI_FALSE(0)	VI_FALSE

Description

An application can request that a TCP/IP provider enable the use of “keep-alive” packets on TCP connections by turning on this attribute. If a connection is dropped as a result of “keep-alive” packets, the error code VI_ERROR_CONN_LOST is returned.

Related Items

See the VI_ATTR_TCPIP_NODELAY description in this chapter. Also see the *SOCKET Resource* description in Appendix B, *Resources*.

VI_ATTR_TCPIP_NODELAY

Resource Classes

TCPIP SOCKET

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE(1) VI_FALSE(0)	VI_TRUE

Description

The Nagle algorithm is disabled when this attribute is enabled (and vice versa). The Nagle algorithm improves network performance by buffering “send” data until a full-size packet can be sent. This attribute is enabled by default in VISA to verify that synchronous writes get flushed immediately.

Related Items

See the VI_ATTR_TCPIP_KEEPALIVE description in this chapter. Also see the *SOCKET Resource* description in Appendix B, *Resources*.

VI_ATTR_TCPIP_PORT

Resource Classes

TCPIP SOCKET

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViUInt16`	0 to FFFFh	N/A

Description

This specifies the port number for a given TCPIP address. For a TCPIP SOCKET Resource, this is a required part of the address string.

Related Items

See the *SOCKET Resource* description in Appendix B, *Resources*.

VI_ATTR_TERMCHAR

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt8	0 to FFh	0Ah (linefeed)

Description

VI_ATTR_TERMCHAR is the termination character. When the termination character is read and VI_ATTR_TERMCHAR_EN is enabled during a read operation, the read operation terminates.

Related Items

See the VI_ATTR_TERMCHAR_EN description in this chapter. Also see the *INSTR Resource*, *INTFC Resource*, *SOCKET Resource* and *SERVANT Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_TERMCHAR_EN

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE(1) VI_FALSE(0)	VI_FALSE

Description

VI_ATTR_TERMCHAR_EN is a flag that determines whether the read operation should terminate when a termination character is received. This attribute is valid for both raw I/O (viRead) and formatted I/O (viScanf).

Related Items

See the VI_ATTR_TERMCHAR description in this chapter. Also see the *INSTR Resource*, *INTFC Resource*, *SOCKET Resource* and *SERVANT Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_TMO_VALUE

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	`ViUInt32`	`VI_TMO_IMMEDIATE(0)`; 1 to FFFFFFFEh; `VI_TMO_INFINITE` (FFFFFFFFh)	2000

Description

`VI_ATTR_TMO_VALUE` specifies the minimum timeout value to use (in milliseconds) when accessing the device associated with the given session. A timeout value of `VI_TMO_IMMEDIATE` means that operations should never wait for the device to respond. A timeout value of `VI_TMO_INFINITE` disables the timeout mechanism.

Notice that the actual timeout value used by the driver may be higher than the requested one. The actual timeout value is returned when this attribute is retrieved via `viGetAttribute()`.

Related Items

See the *INSTR Resource*, *MEMACC Resource*, *INTFC Resource*, *BACKPLANE Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_TRIG_ID

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI BACKPLANE, GPIB-VXI INSTR, GPIB-VXI MEMACC, TCPIP INSTR, VXI BACKPLANE, VXI INSTR, VXI MEMACC, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViInt16	**GPIB**, **Serial, TCPIP**: VI_TRIG_SW(-1)	VI_TRIG_SW
		VXI, **GPIB-VXI**: VI_TRIG_SW(-1); VI_TRIG_TTL0(0) to VI_TRIG_TTL7(7); VI_TRIG_ECL0(8) to VI_TRIG_ECL1(9)	VI_TRIG_SW

Description

VI_ATTR_TRIG_ID is the identifier for the current triggering mechanism.

VI_ATTR_TRIG_ID is Read/Write when the corresponding session is not enabled to receive trigger events. When the session is enabled to receive trigger events, the attribute VI_ATTR_TRIG_ID is Read Only.

Related Items

See the VI_ATTR_RECV_TRIG_ID description in this chapter and viAssertTrigger() in Chapter 5, *Operations*. Also see the *INTFC Resource*, *BACKPLANE Resource*, *INSTR Resource*, and *SERVANT Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_USER_DATA

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, Serial INSTR, PXI INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI BACKPLANE, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	`ViAddr`	0h to FFFFFFFFh	N/A

Description

`VI_ATTR_USER_DATA` is the data used privately by the application for a particular session. This data is not used by VISA for any purposes. It is provided to the application for its own use.

Related Items

See the *VISA Resource Template* description in Appendix B, *Resources*.

VI_ATTR_VXI_DEV_CLASS

Resource Classes

GPIB-VXI INSTR, VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	VI_VXI_CLASS_MEMORY(0) VI_VXI_CLASS_EXTENDED(1) VI_VXI_CLASS_MESSAGE(2) VI_VXI_CLASS_REGISTER(3) VI_VXI_CLASS_OTHER(4)	N/A

Description

This attribute represents the VXI-defined device class to which the resource belongs, either message based (VI_VXI_CLASS_MESSAGE), register based (VI_VXI_CLASS_REGISTER), extended (VI_VXI_CLASS_EXTENDED), or memory (VI_VXI_CLASS_MEMORY). VME devices are usually either register based or belong to a miscellaneous class (VI_VXI_CLASS_OTHER).

Related Items

See *INSTR Resource* in Appendix B, *Resources*.

VI_ATTR_VXI_LA

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, VXI INSTR, VXI MEMACC, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViInt16`	0 to 511	N/A

Description

For an INSTR session, `VI_ATTR_VXI_LA` specifies the logical address of the VXI or VME device used by the given session. For a MEMACC or SERVANT session, this attribute specifies the logical address of the local controller.

Related Items

See the *INSTR Resource*, *MEMACC Resource* and *SERVANT Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_VXI_TRIG_STATUS

Resource Classes

GPIB-VXI BACKPLANE, VXI BACKPLANE

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt32	N/A	N/A

Description

This attribute shows the current state of the VXI trigger lines. This is a bit vector with bits 0-9 corresponding to VI_TRIG_TTL0 through VI_TRIG_ECL1.

Related Items

See VI_ATTR_VXI_VME_INTR_STATUS, VI_ATTR_VXI_TRIG_SUPPORT, and VI_ATTR_VXI_VME_SYSFAIL_STATE in this chapter. See also the *BACKPLANE Resource* description in Appendix B, *Resources*.

VI_ATTR_VXI_TRIG_SUPPORT

Resource Classes

GPIB-VXI INSTR, GPIB-VXI BACKPLANE, VXI INSTR, VXI BACKPLANE

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	`ViUInt32`	N/A	N/A

Description

This attribute shows which VXI trigger lines this implementation supports. This is a bit vector with bits 0-9 corresponding to `VI_TRIG_TTL0` through `VI_TRIG_ECL1`.

Related Items

See the *BACKPLANE Resource* and *INSTR Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_VXI_VME_INTR_STATUS

Resource Classes

GPIB-VXI BACKPLANE, VXI BACKPLANE

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	N/A	N/A

Description

This attribute shows the current state of the VXI/VME interrupt lines. This is a bit vector with bits 0-6 corresponding to interrupt lines 1-7.

Related Items

See VI_ATTR_VXI_TRIG_STATUS and VI_ATTR_VXI_VME_SYSFAIL_STATE in this chapter. See also the *BACKPLANE Resource* description in Appendix B, *Resources*.

VI_ATTR_VXI_VME_SYSFAIL_STATE

Resource Classes

GPIB-VXI BACKPLANE, VXI BACKPLANE

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED(1) VI_STATE_DEASSERTED(0) VI_STATE_UNKNOWN(-1)	N/A

Description

This attribute shows the current state of the VXI/VME SYSFAIL (SYStem FAILure) backplane line.

Related Items

See VI_ATTR_VXI_TRIG_STATUS and VI_ATTR_VXI_VME_INTR_STATUS in this chapter. See also the *BACKPLANE Resource* description in Appendix B, *Resources*.

VI_ATTR_WIN_ACCESS

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, PXI INSTR, VXI INSTR, VXI MEMACC

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Local	ViUInt16	VI_NMAPPED(1) VI_USE_OPERS(2) VI_DEREF_ADDR(3)	VI_NMAPPED

Description

VI_ATTR_WIN_ACCESS specifies the modes in which the current window may be accessed.

- If VI_NMAPPED, the window is not currently mapped.
- If VI_USE_OPERS, the window is accessible through the viPeek*XX*() and viPoke*XX*() operations only.
- If VI_DEREF_ADDR, you can either use operations or directly dereference the mapped address as a pointer.

Related Items

See the VI_ATTR_WIN_ACCESS_PRIV, VI_ATTR_WIN_BASE_ADDR, VI_ATTR_WIN_BYTE_ORDER, and VI_ATTR_WIN_SIZE descriptions in this chapter. See the viMapAddress(), viPeek8/viPeek16/viPeek32, and viPoke8/viPoke16/viPoke32 descriptions in Chapter 5, *Operations*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_WIN_ACCESS_PRIV

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, VXI INSTR, VXI MEMACC

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_DATA_PRIV(0) VI_DATA_NPRIV(1) VI_PROG_PRIV(2) VI_PROG_NPRIV(3) VI_BLCK_PRIV(4) VI_BLCK_NPRIV(5)	VI_DATA_PRIV

Description

VI_ATTR_WIN_ACCESS_PRIV specifies the address modifier to be used in low-level access operations, such as viMapAddress(), viPeek*XX*(), and viPoke*XX*(), when accessing the mapped window.

This attribute is read/write when the corresponding session is not mapped (that is, when VI_ATTR_WIN_ACCESS is VI_NMAPPED). When the session is mapped, this attribute is read only.

Related Items

See the VI_ATTR_DEST_ACCESS_PRIV, VI_ATTR_SRC_ACCESS_PRIV, VI_ATTR_WIN_ACCESS, VI_ATTR_WIN_BASE_ADDR, VI_ATTR_WIN_BYTE_ORDER, and VI_ATTR_WIN_SIZE descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_WIN_BASE_ADDR

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, PXI INSTR, VXI INSTR, VXI MEMACC

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Local	ViBusAddress	0h to FFFFFFFFh	N/A

Description

VI_ATTR_WIN_BASE_ADDR specifies the base address of the interface bus to which this window is mapped. If the value of VI_ATTR_WIN_ACCESS is VI_NMAPPED, the value of this attribute is meaningless.

Related Items

See the VI_ATTR_WIN_ACCESS, VI_ATTR_WIN_ACCESS_PRIV, VI_ATTR_WIN_BYTE_ORDER, and VI_ATTR_WIN_SIZE descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_WIN_BYTE_ORDER

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, VXI INSTR, VXI MEMACC

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_BIG_ENDIAN(0) VI_LITTLE_ENDIAN(1)	VI_BIG_ENDIAN

Description

VI_ATTR_WIN_BYTE_ORDER specifies the byte order to be used in low-level access operations, such as viMapAddress(), viPeek*XX*(), and viPoke*XX*(), when accessing the mapped window.

This attribute is read/write when the corresponding session is not mapped (that is, when VI_ATTR_WIN_ACCESS is VI_NMAPPED). When the session is mapped, this attribute is read only.

Related Items

See the VI_ATTR_DEST_BYTE_ORDER, VI_ATTR_SRC_BYTE_ORDER, VI_ATTR_WIN_ACCESS, VI_ATTR_WIN_ACCESS_PRIV, VI_ATTR_WIN_BASE_ADDR, and VI_ATTR_WIN_SIZE descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_WIN_SIZE

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, PXI INSTR, VXI INSTR, VXI MEMACC

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Local	ViBusSize	0h to FFFFFFFFh	N/A

Description

VI_ATTR_WIN_SIZE specifies the size of the region mapped to this window. If the value of VI_ATTR_WIN_ACCESS is VI_NMAPPED, the value of this attribute is meaningless.

Related Items

See the VI_ATTR_WIN_ACCESS, VI_ATTR_WIN_ACCESS_PRIV, VI_ATTR_WIN_BASE_ADDR, and VI_ATTR_WIN_BYTE_ORDER descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

VI_ATTR_WR_BUF_OPER_MODE

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_FLUSH_ON_ACCESS(1) VI_FLUSH_WHEN_FULL(2)	VI_FLUSH_WHEN_FULL

Description

VI_ATTR_WR_BUF_OPER_MODE specifies the operational mode of the formatted I/O write buffer. When the operational mode is set to VI_FLUSH_WHEN_FULL (default), the buffer is flushed when an END indicator is written to the buffer, or when the buffer fills up. If the operational mode is set to VI_FLUSH_ON_ACCESS, the write buffer is flushed under the same conditions, and also every time a viPrintf() (or related) operation completes.

Related Items

See the VI_ATTR_RD_BUF_OPER_MODE description in this chapter. See the viPrintf() and viFlush() descriptions in Chapter 5, *Operations*. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

4

Events

This chapter describes the VISA events. The event descriptions are listed in alphabetical order for easy reference.

Each event description contains a list below the title indicating the supported resource classes, such as GPIB, Serial, etc. The event description contains a brief description of the event attributes. Chapter 3, *Attributes*, contains more detailed descriptions of the event attributes.

VI_EVENT_CLEAR

Resource Classes

GPIB INTFC, GPIB SERVANT, VXI SERVANT

Description

Notification that the local controller has been sent a device clear message.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.

Related Items

See VI_ATTR_EVENT_TYPE in Chapter 3, *Attributes*. Also see the *INTFC Resource* and *SERVANT Resource* descriptions in Appendix B, *Resources*.

VI_EVENT_EXCEPTION

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Description

This event notifies the application that an error condition has occurred during an operation invocation. In VISA, exceptions are defined as events. The exception-handling model follows the event-handling model for callbacks, and is like any other event in VISA, except that the queueing and suspended handler mechanisms are not allowed.

A VISA operation generating an exception blocks until the exception handler execution is completed. However, an exception handler sometimes may prefer to terminate the program prematurely without returning the control to the operation generating the exception. VISA does not preclude an application from using a platform-specific or language-specific exception handling mechanism from within the VISA exception handler. For example, the C++ try/catch block can be used in an application in conjunction with the C++ throw mechanism from within the VISA exception handler.

When using the C++ try/catch/throw or other exception-handling mechanisms, the control will not return to the VISA system. This has some important repercussions:

- If multiple handlers were installed on the exception event, the handlers that were not invoked prior to the current handler will not be invoked for the current exception.
- The exception context will not be deleted by the VISA system when a C++ exception is used. In this case, the application should delete the exception context as soon as the application has no more use for the context, before terminating the session. An application should use the `viClose()` operation to delete the exception context.

One situation in which an exception event will not be generated is in the case of asynchronous operations. If the error is detected after the operation is posted—once the asynchronous portion has begun—the status is returned normally via the I/O completion event. However, if an error occurs before the asynchronous portion begins—the error is returned from the asynchronous operation itself—then the exception event will still be raised. This deviation is due to the fact that asynchronous operations already raise an event when they complete, and this I/O completion event may occur in the context of a separate thread previously unknown to the application. In summary, a single application event handler can easily handle error conditions arising from both exception events and failed asynchronous operations.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event. This attribute always has the value of VI_EVENT_EXCEPTION for this event type.
VI_ATTR_STATUS	Contains the status code returned by the operation generating the error.
VI_ATTR_OPER_NAME	Contains the name of the operation generating the event.

Related Items

See the VI_ATTR_EVENT_TYPE, VI_ATTR_STATUS, and VI_ATTR_OPER_NAME descriptions in Chapter 3, *Attributes*. See the viEnableEvent() description in Chapter 5, *Operations*. Also see Appendix B, *Resources*.

VI_EVENT_GPIB_CIC

Resource Classes

GPIB INTFC

Description

Notification that the GPIB controller has gained or lost CIC (controller in charge) status.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.
VI_ATTR_GPIB_RECV_CIC_STATE	Specifies whether the CIC status was gained or lost.

Related Items

See VI_ATTR_EVENT_TYPE and VI_ATTR_GPIB_CIC_STATE in Chapter 3, *Attributes*. Also see the *INTFC Resource* description in Appendix B, *Resources*.

VI_EVENT_GPIB_LISTEN

Resource Classes

GPIB INTFC, GPIB SERVANT

Description

Notification that the GPIB controller has been addressed to listen.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.

Related Items

See VI_ATTR_EVENT_TYPE in Chapter 3, *Attributes*. Also see the *INTFC Resource* and *SERVANT Resource* descriptions in Appendix B, *Resources*.

VI_EVENT_GPIB_TALK

Resource Classes

GPIB INTFC, GPIB SERVANT

Description

Notification that the GPIB controller has been addressed to talk.

Event Attribute

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.

Related Items

See VI_ATTR_EVENT_TYPE in Chapter 3, *Attributes*. Also see the *INTFC Resource* and *SERVANT Resource* descriptions in Appendix B, *Resources*.

VI_EVENT_IO_COMPLETION

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI SERVANT

Description

This event notifies the application that an asynchronous operation has completed.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event. This attribute always has the value of VI_EVENT_IO_COMPLETION for this event type.
VI_ATTR_STATUS	Contains the return code of the asynchronous I/O operation that has completed.
VI_ATTR_JOB_ID	Contains the job ID of the asynchronous operation that has completed.
VI_ATTR_BUFFER	Contains the address of the buffer that was used in the asynchronous operation.
VI_ATTR_RET_COUNT	Contains the actual number of elements that were asynchronously transferred.
VI_ATTR_OPER_NAME	Contains the name of the operation generating the event.

Related Items

See the VI_ATTR_EVENT_TYPE, VI_ATTR_STATUS, VI_ATTR_JOB_ID, VI_ATTR_BUFFER, VI_ATTR_RET_COUNT, and VI_ATTR_OPER_NAME descriptions in Chapter 3, *Attributes*. Also see the *INSTR Resource*, *MEMACC Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

VI_EVENT_PXI_INTR

Resource Classes

PXI INSTR

Description

This event notifies that a PXI interrupt has occurred.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.

Related Items

See the VI_ATTR_EVENT_TYPE description in Chapter 3, *Attributes*. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_EVENT_SERVICE_REQ

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB-VXI INSTR, TCPIP INSTR, VXI INSTR

Description

This event notifies the application that a service request was received from the device or interface associated with the given session.

Note When you receive a `VI_EVENT_SERVICE_REQ` on an INSTR session, you must call `viReadSTB()` to guarantee delivery of future service request events on the given session.

Event Attributes

Symbolic Name	Description
`VI_ATTR_EVENT_TYPE`	Unique logical identifier of the event. This attribute always has the value of `VI_EVENT_SERVICE_REQ` for this event type.

Related Items

See the `VI_ATTR_EVENT_TYPE` description in Chapter 3, *Attributes*. See the `viReadSTB()` description in Chapter 5, *Operations*. Also see the *INSTR Resource* and *INTFC Resource* descriptions in Appendix B, *Resources*.

VI_EVENT_TRIG

Resource Classes

GPIB INTFC, GPIB SERVANT, VXI INSTR, VXI BACKPLANE, VXI SERVANT

Description

This event notifies the application that a trigger interrupt was received from the device. This may be either a hardware or software trigger, depending on the interface and the current session settings.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event. This attribute always has the value of VI_EVENT_TRIG for this event type.
VI_ATTR_RECV_TRIG_ID	The identifier of the triggering mechanism on which the specified trigger event was received.

Related Items

See the VI_ATTR_TRIG_ID, VI_ATTR_EVENT_TYPE, and VI_ATTR_RECV_TRIG_ID descriptions in Chapter 3, *Attributes*. Also see the *INSTR Resource*, *INTFC Resource*, *BACKPLANE Resource* and *SERVANT Resource* descriptions in Appendix B, *Resources*.

VI_EVENT_VXI_SIGP

Resource Classes

VXI INSTR

Description

This event notifies the application that a VXIbus signal or VXIbus interrupt was received from the device associated with the given session.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event. This attribute always has the value of VI_EVENT_VXI_SIGP for this event type.
VI_ATTR_SIGP_STATUS_ID	The 16-bit Status/ID value retrieved during the IACK cycle or from the Signal register.

Related Items

See the VI_ATTR_EVENT_TYPE and VI_ATTR_SIGP_STATUS_ID descriptions in Chapter 3, *Attributes*. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_EVENT_VXI_VME_INTR

Resource Classes

VXI INSTR

Description

This event notifies the application that a VXIbus interrupt was received from the device associated with the given session.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event. This attribute always has the value of VI_EVENT_VXI_VME_INTR for this event type.
VI_ATTR_INTR_STATUS_ID	The 32-bit Status/ID value retrieved during the IACK cycle.
VI_ATTR_RECV_INTR_LEVEL	The VXI interrupt level on which the interrupt was received.

Related Items

See the VI_ATTR_EVENT_TYPE, VI_ATTR_INTR_STATUS_ID, and VI_ATTR_RECV_INTR_LEVEL descriptions in Chapter 3, *Attributes*. Also see the *INSTR Resource* description in Appendix B, *Resources*.

VI_EVENT_VXI_VME_SYSFAIL

Resource Classes

VXI BACKPLANE

Description

Notification that the VXI/VME SYSFAIL* line has been asserted.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.

Related Items

See VI_ATTR_EVENT_TYPE in Chapter 3, *Attributes*. Also see *BACKPLANE Resource* in Appendix B, *Resources*.

VI_EVENT_VXI_VME_SYSRESET

Resource Classes

VXI BACKPLANE, VXI SERVANT

Description

Notification that the VXI/VME SYSRESET* line has been asserted.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.

Related Items

See VI_ATTR_EVENT_TYPE in Chapter 3, *Attributes*. Also see *BACKPLANE Resource* and *SERVANT Resource* in Appendix B, *Resources*.

5

Operations

This chapter describes the VISA operations. The operation descriptions are listed in alphabetical order for easy reference.

Each event description contains a brief *Purpose* statement below the title. You will then see the operation defined in both ANSI C and Visual Basic version 4 syntax, with the parameters set in **boldface** type. A list indicating the supported resource classes, such as GPIB, Serial, etc. is followed by a table that describes each parameter and indicates whether it is an input or output parameter (or both, in some cases). The *Return Values* section describes the completion and error codes, followed by a detailed *Description* section. The *Related Items* section directs you toward related operations, attributes, events, or resource descriptions. If you want to know specifically about attributes, events, and operations of the INSTR Resource, for example, you should turn to the *INSTR Resource* section in Appendix B, *Resources*.

viAssertIntrSignal

Purpose

Asserts the specified interrupt or signal.

C Syntax

```
ViStatus viAssertIntrSignal(ViSession vi, ViInt16 mode,
ViUInt32 statusID)
```

Visual Basic Syntax

```
viAssertIntrSignal&(ByVal vi&, ByVal mode%, ByVal statusID&)
```

Resource Classes

GPIB-VXI BACKPLANE, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
mode	IN	This specifies how to assert the interrupt. See the Description section for actual values.
statusID	IN	This is the status value to be presented during an interrupt acknowledge cycle. See the *Description* section for valid values.

Return Values

Completion Code	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.

Error Codes	Description
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INTR_PENDING	An interrupt is still pending from a previous call.
VI_ERROR_INV_MODE	The value specified by the **mode** parameter is invalid.
VI_ERROR_NSUP_INTR	The interface cannot generate an interrupt on the requested level or with the requested **statusID** value.
VI_ERROR_NSUP_MODE	The specified **mode** is not supported by this VISA implementation.

Description

This operation can be used to assert a device interrupt condition. In VXI, for example, this can be done with either a VXI signal or a VXI interrupt. On certain bus types, the statusID parameter may be ignored. The following table lists the valid values for the **mode** parameter.

Mode	Action Description
VI_ASSERT_USE_ASSIGNED	Use whatever notification method that has been assigned to the local device.
VI_ASSERT_SIGNAL	Send the notification via a VXI signal.
VI_ASSERT_IRQ1 - VI_ASSERT_IRQ7	Send the interrupt via the specified VXI/VME IRQ line. This uses the standard VXI/VME ROAK (release on acknowledge) interrupt mechanism, rather than the older VME RORA (release on register access) mechanism.

Related Items

See `viAssertUtilSignal( )` in this chapter and the *BACKPLANE Resource* and *SERVANT Resource* descriptions in Appendix B, *Resources*.

viAssertTrigger

Purpose

Asserts software or hardware trigger.

C Syntax

```
ViStatus viAssertTrigger(ViSession vi, ViUInt16 protocol)
```

Visual Basic Syntax

```
viAssertTrigger&(ByVal vi&, ByVal protocol%)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB-VXI INSTR, GPIB-VXI BACKPLANE, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI BACKPLANE

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
protocol	IN	Trigger protocol to use during assertion. Valid values are: VI_TRIG_PROT_DEFAULT (0), VI_TRIG_PROT_ON (1), VI_TRIG_PROT_OFF (2), and VI_TRIG_PROT_SYNC (5).

Return Values

Completion Codes	Description
VI_SUCCESS	The specified trigger was successfully asserted to the device.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.

Error Codes	Description
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_INV_PROT	The protocol specified is invalid.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_INP_PROT_VIOL	Device reported an input protocol error during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_LINE_IN_USE	The specified trigger line is currently in use.
VI_ERROR_NCIC	The interface associated with the given **vi** is not currently the controller in charge.
VI_ERROR_NLISTENERS	No-listeners condition is detected (both NRFD and NDAC are unasserted).
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_CONN_LOST	The I/O connection for the given session has been lost.

Description

The viAssertTrigger() operation will source a software or hardware trigger dependent on the interface type. For a GPIB device, the device is addressed to listen, and then the GPIB *GET* command is sent. For a VXI device, if VI_ATTR_TRIG_ID is VI_TRIG_SW, then the device is sent the Word Serial *Trigger* command; for any other values of the attribute, a hardware trigger is sent on the line that corresponds to the value of that attribute. For a session to a Serial device or Ethernet socket, if VI_ATTR_IO_PROT is VI_PROT_4882_STRS, the device is sent the string "*TRG\n"; otherwise, this operation is not valid.

For GPIB, serial, and VXI software triggers, VI_TRIG_PROT_DEFAULT is the only valid protocol. For VXI hardware triggers, VI_TRIG_PROT_DEFAULT is equivalent to VI_TRIG_PROT_SYNC.

Related Items

See the VI_ATTR_TRIG_ID description in Chapter 3, *Attributes*. Also see the *INSTR Resource*, *SOCKET Resource*, *BACKPLANE Resource*, and *INTFC Resource* descriptions in Appendix B, *Resources*.

viAssertUtilSignal

Purpose

Asserts or deasserts the specified utility bus signal.

C Syntax

```
viStatus viAssertUtilSignal(ViSession vi, ViUInt16 line)
```

Visual Basic Syntax

```
viAssertUtilSignal& (ByVal vi&, ByVal line%)
```

Resource Classes

GPIB-VXI BACKPLANE, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
line	IN	Specifies the utility bus signal to assert. This can be the value VI_UTIL_ASSERT_SYSRESET, VI_UTIL_ASSERT_SYSFAIL, or VI_UTIL_DEASSERT_SYSFAIL.

Return Values

Completion Code	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.

Error Codes	Description
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_INV_LINE	The value specified by the **line** parameter is invalid.

Description

This operation can be used to assert either the SYSFAIL or SYSRESET utility bus interrupts on the VXIbus backplane. This operation is valid only on BACKPLANE (mainframe) and VXI SERVANT (servant) sessions.

Asserting SYSRESET (also known as HARD RESET in the VXI specification) should be used only when it is necessary to promptly terminate operation of all devices in a VXIbus system. This is a serious action that always affects the entire VXIbus system.

Related Items

See `viAssertIntrSignal( )` in this chapter and the *BACKPLANE Resource* and *SERVANT Resource* descriptions in Appendix B, *Resources*.

viBufRead

Purpose

Reads data from device or interface through the use of a formatted I/O read buffer.

C Syntax

```
ViStatus viBufRead(ViSession vi, ViPBuf buf, ViUInt32 count,
ViPUInt32 retCount)
```

Visual Basic Syntax

```
viBufRead&(ByVal vi&, ByVal buf$, ByVal count&, retCount&)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	OUT	Location of a buffer to receive data from device.
count	IN	Number of bytes to be read.
retCount	OUT	Number of bytes actually transferred.

Return Values

Completion Codes	Description
VI_SUCCESS	The operation completed successfully and the END indicator was received (for interfaces that have END indicators). This completion code is returned regardless of whether the termination character is received or the number of bytes read is equal to **count**.
VI_SUCCESS_TERM_CHAR	The specified termination character was read but no END indicator was received. This completion code is returned regardless of whether the number of bytes read is equal to **count**.
VI_SUCCESS_MAX_CNT	The number of bytes read is equal to **count**. No END indicator was received and no termination character was read.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_IO	An unknown I/O error occurred during transfer.

Description

The `viBufRead()` operation is similar to `viRead()` and does not perform any kind of data formatting. It differs from `viRead()` in that the data is read from the formatted I/O read buffer—the same buffer used by `viScanf()` and related operations—rather than directly from the device. You can intermix this operation with `viScanf()`, but you should not mix it with `viRead()`.

`VI_NULL` is a special value for the **retCount** parameter. If you pass `VI_NULL` for **retCount**, the number of bytes transferred is not returned. You may find this useful if you need to know only whether the operation succeeded or failed.

Related Items

See the `viRead()` and `viBufWrite()` descriptions in this chapter. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viBufWrite

Purpose

Writes data to a formatted I/O write buffer synchronously.

C Syntax

```
ViStatus viBufWrite(ViSession vi, ViBuf buf, ViUInt32 count,
ViPUInt32 retCount)
```

Visual Basic Syntax

```
viBufWrite&(ByVal vi&, ByVal buf$, ByVal count&, retCount&)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	IN	Location of a block of data.
count	IN	Number of bytes to be written.
retCount	OUT	Number of bytes actually transferred.

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.

Error Codes	Description
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_INV_SETUP	Unable to start write operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_IO	An unknown I/O error occurred during transfer.

Description

The viBufWrite() operation is similar to viWrite() and does not perform any kind of data formatting. It differs from viWrite() in that the data is written to the formatted I/O write buffer—the same buffer used by viPrintf() and related operations—rather than directly to the device. You can intermix this operation with viPrintf(), but you should not mix it with viWrite().

If this operation returns VI_ERROR_TMO, the write buffer for the specified session is cleared.

VI_NULL is a special value for the **retCount** parameter. If you pass VI_NULL for **retCount**, the number of bytes transferred is not returned. You may find this useful if you need to know only whether the operation succeeded or failed.

Related Items

See the viWrite() and viBufRead() descriptions in this chapter. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viClear

Purpose

Clears a device.

C Syntax

```
ViStatus viClear(ViSession vi)
```

Visual Basic Syntax

```
viClear&(ByVal vi&)
```

Resource Classes

GPIB INSTR, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.

Error Codes	Description
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_NCIC	The interface associated with the given **vi** is not currently the controller in charge.
VI_ERROR_NLISTENERS	No-listeners condition is detected (both NRFD and NDAC are unasserted).
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_CONN_LOST	The I/O connection for the given session has been lost.

Description

The viClear() operation performs an IEEE 488.1-style clear of the device (for VXI, the Word Serial Clear command is used; for GPIB systems, the Selected Device Clear command is used). For a session to a Serial device or Ethernet socket, if VI_ATTR_IO_PROT is VI_PROT_4882_STRS, the device is sent the string "*CLS\n"; otherwise, this operation is not valid. Invoking viClear() on an INSTR Resource will also discard the read and write buffers used by the formatted I/O services for that session.

Related Items

See the *INSTR Resource* and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viClose

Purpose

Closes the specified session, event, or find list.

C Syntax

```
ViStatus viClose(ViObject vi)
```

Visual Basic Syntax

```
viClose&(ByVal vi&)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session, event, or find list.

Return Values

Completion Codes	Description
VI_SUCCESS	Session closed successfully.
VI_WARN_NULL_OBJECT	The specified object reference is uninitialized.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given object reference is invalid.
VI_ERROR_CLOSING_FAILED	Unable to deallocate the previously allocated data structures corresponding to this session or object reference.

Description

The `viClose()` operation closes a session, event, or a find list. In this process all the data structures that had been allocated for the specified **vi** are freed. Calling `viClose()` on a VISA Resource Manager session will also close all I/O sessions associated with that resource manager session.

Related Items

See the `viOpen()`, `viOpenDefaultRM()`, `viFindRsrc()`, and `viWaitOnEvent()` descriptions in this chapter. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

viDisableEvent

Purpose

Disables notification of the specified event type(s) via the specified mechanism(s).

C Syntax

```
ViStatus viDisableEvent(ViSession vi, ViEventType eventType,
ViUInt16 mechanism)
```

Visual Basic Syntax

```
viDisableEvent&(ByVal vi&, ByVal eventType&, ByVal mechanism%)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
eventType	IN	Logical event identifier.
mechanism	IN	Specifies event handling mechanisms to be disabled. The queuing mechanism is disabled by specifying `VI_QUEUE` (1), and the callback mechanism is disabled by specifying `VI_HNDLR` (2) or `VI_SUSPEND_HNDLR` (4). It is possible to disable both mechanisms simultaneously by specifying `VI_ALL_MECH` (FFFFh).

Return Values

Completion Codes	Description
`VI_SUCCESS`	Event disabled successfully.
`VI_SUCCESS_EVENT_DIS`	Specified event is already disabled for at least one of the specified mechanisms.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_INV_EVENT	Specified **eventType** is not supported by the resource.
VI_ERROR_INV_MECH	Invalid **mechanism** specified.

Description

The `viDisableEvent()` operation disables servicing of an event identified by the **eventType** parameter for the mechanisms specified in the **mechanism** parameter. This operation prevents *new* event occurrences from being added to the queue(s). However, event occurrences already existing in the queue(s) are not flushed. Use `viDiscardEvents()` if you want to discard events remaining in the queue(s).

Specifying `VI_ALL_ENABLED_EVENTS` for the **eventType** parameter allows a session to stop receiving all events. The session can stop receiving queued events by specifying `VI_QUEUE`. Applications can stop receiving callback events by specifying either `VI_HNDLR` or `VI_SUSPEND_HNDLR`. Specifying `VI_ALL_MECH` disables both the queuing and callback mechanisms.

Related Items

See the `viEnableEvent()` description in this chapter. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

viDiscardEvents

Purpose

Discards event occurrences for specified event types and mechanisms in a session.

C Syntax

```
ViStatus viDiscardEvents(ViSession vi, ViEventType eventType,
ViUInt16 mechanism)
```

Visual Basic Syntax

```
viDiscardEvents&(ByVal vi&, ByVal eventType&, ByVal mechanism%)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
eventType	IN	Logical event identifier.
mechanism	IN	Specifies the mechanisms for which the events are to be discarded. The VI_QUEUE (1) value is specified for the queuing mechanism and the VI_SUSPEND_HNDLR (4) value is specified for the pending events in the callback mechanism. It is possible to specify both mechanisms simultaneously by specifying VI_ALL_MECH (FFFFh).

Return Values

Completion Codes	Description
VI_SUCCESS	Event queue flushed successfully.
VI_SUCCESS_QUEUE_EMPTY	Operation completed successfully, but queue was already empty.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_INV_EVENT	Specified **eventType** is not supported by the resource.
VI_ERROR_INV_MECH	Invalid **mechanism** specified.

Description

The viDiscardEvents() operation discards all pending occurrences of the specified event types and mechanisms from the specified session. Specifying VI_ALL_ENABLED_EVENTS for the **eventType** parameter discards events of every type that is enabled for the given session. The information about all the event occurrences which have not yet been handled is discarded. This operation is useful to remove event occurrences that an application no longer needs. The discarded event occurrences are not available to a session at a later time. This operation does not apply to event contexts that have already been delivered to the application.

Related Items

See the viEnableEvent(), viDisableEvent(), and viWaitOnEvent() descriptions in this chapter. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

viEnableEvent

Purpose

Enables notification of a specified event.

C Syntax

```
ViStatus viEnableEvent(ViSession vi, ViEventType eventType,
ViUInt16 mechanism, ViEventFilter context)
```

Visual Basic Syntax

```
viEnableEvent&(ByVal vi&, ByVal eventType&, ByVal mechanism%,
ByVal context&)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
eventType	IN	Logical event identifier.
mechanism	IN	Specifies event handling mechanisms to be enabled. The queuing mechanism is enabled by specifying `VI_QUEUE` (1), and the callback mechanism is enabled by specifying `VI_HNDLR` (2) or `VI_SUSPEND_HNDLR` (4). It is possible to enable both mechanisms simultaneously by specifying *bit-wise OR* of `VI_QUEUE` and one of the two mode values for the callback mechanism.
context	IN	`VI_NULL` (0).

Return Values

Completion Codes	Description
VI_SUCCESS	Event enabled successfully.
VI_SUCCESS_EVENT_EN	Specified event is already enabled for at least one of the specified mechanisms.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_INV_EVENT	Specified **eventType** is not supported by the resource.
VI_ERROR_INV_MECH	Invalid **mechanism** specified for the event.
VI_ERROR_INV_CONTEXT	Specified event context is invalid.
VI_ERROR_HNDLR_NINSTALLED	A handler is not currently installed for the specified event. The session cannot be enabled for the VI_HNDLR mode of the callback mechanism.
VI_ERROR_NSUP_MECH	The specified **mechanism** is not supported for the given **eventType**.

Description

The viEnableEvent() operation enables notification of an event identified by the **eventType** parameter for mechanisms specified in the **mechanism** parameter. The specified session can be enabled to queue events by specifying VI_QUEUE. Applications can enable the session to invoke a callback function to execute the handler by specifying VI_HNDLR. The applications are required to install at least one handler to be enabled for this mode. Specifying VI_SUSPEND_HNDLR enables the session to receive callbacks, but the invocation of the handler is deferred to a later time. Successive calls to this operation replace the old callback mechanism with the new callback mechanism. Specifying VI_ALL_ENABLED_EVENTS for the **eventType** parameter refers to all events which have previously been enabled on this session, making it easier to switch between the two callback mechanisms for multiple events.

Related Items

See the `viDisableEvent()` and `viWaitOnEvent()` descriptions in this chapter. Also see the `viInstallHandler()` and `viUninstallHandler()` descriptions in this chapter for information about installing and uninstalling event handlers. See Chapter 4, *Events*, for a list of events that you can enable. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

viEventHandler

Purpose

Event service handler procedure prototype.

C Syntax

```
ViStatus _VI_FUNCH viEventHandler(ViSession vi,
ViEventType eventType, ViEvent context, ViAddr userHandle)
```

Visual Basic Syntax

N/A

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
eventType	IN	Logical event identifier.
context	IN	A handle specifying the unique occurrence of an event.
userHandle	IN	A value specified by an application that can be used for identifying handlers uniquely in a session for an event.

Return Values

Completion Codes	Description
VI_SUCCESS	Event handled successfully.
VI_SUCCESS_NCHAIN	Event handled successfully. Do not invoke any other handlers on this session for this event.

Description

`viEventHandler()` is not an actual VISA operation. Rather, it is the prototype for a user event handler that is installed with the `viInstallHandler()` operation. The user handler is called whenever a session receives an event and is enabled for handling events in the `VI_HNDLR` mode. The handler services the event and returns `VI_SUCCESS` on completion. The VISA system automatically invokes the `viClose()` operation on the event context when a user handler returns.

Because the event context must still be valid after the user handler returns (so that VISA can free it up), an application should not invoke the `viClose()` operation on an event context passed to a user handler.

Note For advanced users—If the user handler will not return to VISA, the application should call `viClose()` on the event context to manually delete the event object. This situation may occur when a handler throws a C++ exception in response to a VISA exception event.

Normally, an application should always return `VI_SUCCESS` from all callback handlers. If a specific handler does not want other handlers to be invoked for the given event for the given session, it should return `VI_SUCCESS_NCHAIN`. No return value from a handler on one session will affect callbacks on other sessions. Future versions of VISA (or specific implementations of VISA) may take actions based on other return values, so a user should return `VI_SUCCESS` from handlers unless there is a specific reason to do otherwise.

Related Items

See `viInstallHandler()` and `viUninstallHandler()` descriptions in this chapter. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

viFindNext

Purpose

Returns the next resource from the list of resources found during a previous call to `viFindRsrc()`.

C Syntax

```
ViStatus viFindNext(ViFindList findList, ViChar instrDesc[])
```

Visual Basic Syntax

```
viFindNext&(ByVal findList&, ByVal instrDesc$)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
findList	IN	Describes a find list. This parameter must be created by `viFindRsrc()`.
instrDesc	OUT	Returns a string identifying the location of a device. Strings can then be passed to `viOpen()` to establish a session to the given device.

Return Values

Completion Codes	Description
VI_SUCCESS	Resource(s) found.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given object reference is invalid.
VI_ERROR_NSUP_OPER	The given **findList** does not support this operation.
VI_ERROR_RSRC_NFOUND	There are no more matches.

Description

The `viFindNext()` operation returns the next device found in the list created by `viFindRsrc()`. The list is referenced by the handle that was returned by `viFindRsrc()`.

Note The size of the **instrDesc** parameter should be at least 256 bytes.

Related Items

See the `viFindRsrc()` description in this chapter. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

viFindRsrc

Purpose

Queries a VISA system to locate the resources associated with a specified interface.

C Syntax

```
ViStatus viFindRsrc(ViSession sesn, ViString expr,
ViPFindList findList, ViPUInt32 retcnt, ViChar instrDesc[])
```

Visual Basic Syntax

```
viFindRsrc&(ByVal sesn&, ByVal expr$, findList&, retcnt&,
ByVal instrDesc$)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
sesn	IN	Resource Manager session (should always be the session returned from `viOpenDefaultRM()`).
expr	IN	This is a regular expression followed by an optional logical expression. Refer to the discussion of the Description String in the *Description* section of this operation.
findList	OUT	Returns a handle identifying this search session. This handle will be used as an input in `viFindNext()`.
retcnt	OUT	Number of matches.
instrDesc	OUT	Returns a string identifying the location of a device. Strings can then be passed to `viOpen()` to establish a session to the given device.

Return Values

Completion Codes	Description
VI_SUCCESS	Resource(s) found.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **sesn** does not support this operation. This operation is supported only by a Resource Manager session.
VI_ERROR_INV_EXPR	Invalid expression specified for search.
VI_ERROR_RSRC_NFOUND	Specified expression does not match any devices.

Description

The `viFindRsrc()` operation matches the value specified in the **expr** parameter with the resources available for a particular interface. A regular expression is a string consisting of ordinary characters as well as special characters. You use a regular expression to specify patterns to match in a given string; in other words, it is a search criterion. The `viFindRsrc()` operation uses a case-insensitive compare feature when matching resource names against the regular expression specified in **expr**. For example, calling `viFindRsrc()` with "`VXI?*INSTR`" would return the same resources as invoking it with "`vxi?*instr`".

On successful completion, this function returns the first resource found in the list and returns a count (**retcnt**) to indicate if there were more resources found for the designated interface. This function also returns, in the **findList** parameter, a handle to a find list. This handle points to the list of resources and it must be used as an input to `viFindNext()`. When this handle is no longer needed, it should be passed to `viClose()`. Notice that **retcnt** and **findList** are optional parameters. This is useful if only the first match is important, and the number of matches is not needed. If you specify `VI_NULL` in the **findList** parameter and the operation completes successfully, VISA automatically invokes `viClose()` on the find list handle rather than returning it to the application.

Note The size of the **instrDesc** parameter should be at least 256 bytes.

The search criteria specified in the **expr** parameter has two parts: a regular expression over a resource string, and an optional logical expression over attribute values. The regular expression is matched against the resource strings of resources known to the VISA Resource Manager. If the resource string matches the regular expression, the attribute values of the

resource are then matched against the expression over attribute values. If the match is successful, the resource has met the search criteria and gets added to the list of resources found. All resource strings returned by `viFindRsrc()` will always be recognized by `viOpen()`. However, `viFindRsrc()` will not necessarily return all strings that you can pass to `viParseRsrc()` or `viOpen()`. This is especially true for network and TCPIP resources.

Special Characters and Operators	Meaning
`?`	Matches any one character.
`\`	Makes the character that follows it an ordinary character instead of special character. For example, when a question mark follows a backslash (`\?`), it matches the `?` character instead of any one character.
`[list]`	Matches any one character from the enclosed `list`. You can use a hyphen to match a range of characters.
`[^list]`	Matches any character not in the enclosed `list`. You can use a hyphen to match a range of characters.
`*`	Matches 0 or more occurrences of the preceding character or expression.
`+`	Matches 1 or more occurrences of the preceding character or expression.
`exp|exp`	Matches either the preceding or following expression. The or operator `|` matches the entire expression that precedes or follows it and not just the character that precedes or follows it. For example, `VXI|GPIB` means `(VXI)|(GPIB)`, not `VX(I|G)PIB`.
`(exp)`	Grouping characters or expressions.

Regular Expression	Sample Matches
GPIB?*INSTR	Matches GPIB0::2::INSTR, GPIB1::1::1::INSTR, and GPIB-VXI1::8::INSTR.
GPIB[0-9]*::?*INSTR	Matches GPIB0::2::INSTR and GPIB1::1::1::INSTR but not GPIB-VXI1::8::INSTR.
GPIB[^0]::?*INSTR	Matches GPIB1::1::1::INSTR but not GPIB0::2::INSTR or GPIB12::8::INSTR.
VXI?*INSTR	Matches VXI0::1::INSTR but not GPIB-VXI0::1::INSTR.
GPIB-VXI?*INSTR	Matches GPIB-VXI0::1::INSTR but not VXI0::1::INSTR.
?*VXI[0-9]*::?*INSTR	Matches VXI0::1::INSTR and GPIB-VXI0::1::INSTR.
ASRL[0-9]*::?*INSTR	Matches ASRL1::INSTR but not VXI0::5::INSTR.
ASRL1+::INSTR	Matches ASRL1::INSTR and ASRL11::INSTR but not ASRL2::INSTR.
(GPIB|VXI)?*INSTR	Matches GPIB1::5::INSTR and VXI0::3::INSTR but not ASRL2::INSTR.
(GPIB0|VXI0)::1::INSTR	Matches GPIB0::1::INSTR and VXI0::1::INSTR.
?*INSTR	Matches all INSTR (device) resources.
?*VXI[0-9]*::?*MEMACC	Matches VXI0::MEMACC and GPIB-VXI1::MEMACC.
VXI0::?*	Matches VXI0::1::INSTR, VXI0::2::INSTR, and VXI0::MEMACC.
?*	Matches all resources.

By using the optional attribute expression, you can construct flexible and powerful expressions with the use of logical ANDs (&&), ORs(||), and NOTs (!). You can use equal (==) and unequal (!=) comparators to compare attributes of any type, and other inequality comparators (>, <, >=, <=) to compare attributes of numeric type. Use only

global attributes in the attribute expression. Local attributes are not allowed in the logical expression part of the **expr** parameter.

Expr Parameter	Meaning
`GPIB[0-9]*::?*::?*::INSTR {VI_ATTR_GPIB_SECONDARY_ADDR > 0 && VI_ATTR_GPIB_SECONDARY_ADDR < 10}`	Find all GPIB devices that have secondary addresses from 1 to 9.
`ASRL?*INSTR{VI_ATTR_ASRL_BAUD == 9600}`	Find all serial ports configured at 9600 baud.
`?*VXI?INSTR{VI_ATTR_MANF_ID == 0xFF6 && !(VI_ATTR_VXI_LA ==0 || VI_ATTR_SLOT <= 0)}`	Find all VXI instrument resources having manufacturer ID FF6 and which are not logical address 0, slot 0, or external controllers.

Related Items

See the `viClose()` and `viFindNext()` descriptions in this chapter. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

viFlush

Purpose

Manually flushes the specified buffers associated with formatted I/O operations and/or serial communication.

C Syntax

```
ViStatus viFlush(ViSession vi, ViUInt16 mask)
```

Visual Basic Syntax

```
viFlush&(ByVal vi&, ByVal mask%)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
mask	IN	Specifies the action to be taken with flushing the buffer. Refer to the *Description* section for more information.

Return Values

Completion Codes	Description
VI_SUCCESS	Buffers flushed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_IO	Could not perform read/write operation because of I/O error.

Error Codes	Description
VI_ERROR_TMO	The read/write operation was aborted because timeout expired while operation was in progress.
VI_ERROR_INV_MASK	The specified **mask** does not specify a valid flush operation on read/write resource.

Description

The value of **mask** can be one of the following flags.

Flag	Interpretation
VI_READ_BUF (1)	Discard the read buffer contents. If data was present in the read buffer and no END-indicator was present, read from the device until encountering an END indicator (which causes the loss of data). This action resynchronizes the next viScanf() call to read a <TERMINATED RESPONSE MESSAGE>. (Refer to the IEEE 488.2 standard.)
VI_READ_BUF_DISCARD (4)	Discard the read buffer contents (does not perform any I/O to the device).
VI_WRITE_BUF (2)	Flush the write buffer by writing all buffered data to the device.
VI_WRITE_BUF_DISCARD (8)	Discard the write buffer contents (does not perform any I/O to the device).
VI_IO_IN_BUF (16)	Discard the receive buffer contents (same as VI_IO_IN_BUF_DISCARD).
VI_IO_IN_BUF_DISCARD (64)	Discard the receive buffer contents (does not perform any I/O to the device).
VI_IO_OUT_BUF (32)	Flush the transmit buffer by writing all buffered data to the device.
VI_IO_OUT_BUF_DISCARD (128)	Discard the transmit buffer contents (does not perform any I/O to the device).

It is possible to combine any of these read flags and write flags for different buffers by ORing the flags. However, combining two flags for the same buffer in the same call to `viFlush()` is illegal.

Notice that when using formatted I/O operations with a session to a Serial device or Ethernet socket, a flush of the formatted I/O buffers also causes the corresponding I/O communication buffers to be flushed. For example, calling `viFlush()` with `VI_WRITE_BUF` also flushes the `VI_IO_OUT_BUF`.

In previous versions of VISA, `VI_IO_IN_BUF` was known as `VI_ASRL_IN_BUF` and `VI_IO_OUT_BUF` was known as `VI_ASRL_OUT_BUF`.

Implicit vs. Explicit Flushing

Although you can explicitly flush the buffers by making a call to `viFlush()`, the buffers are flushed implicitly under some conditions. These conditions vary for the `viPrintf()` and `viScanf()` operations.

Flushing a write buffer immediately sends any queued data to the device. The write buffer is maintained by the `viPrintf()` operation. To explicitly flush the write buffer, you can make a call to the `viFlush()` operation with a write flag set. In addition, the write buffer is flushed automatically under the following conditions:

1. When an END-indicator character is sent (that is, the \n character is specified in the formatting string).
2. When the buffer is full.
3. In response to a call to `viSetBuf()` with the `VI_WRITE_BUF` flag set.

Flushing a read buffer discards the data in the read buffer. This guarantees that the next call to a `viScanf()` (or related) operation reads data directly from the device rather than from queued data residing in the read buffer. The read buffer is maintained by the `viScanf()` operation. To explicitly flush the read buffer, you can make a call to the `viFlush()` operation with a read flag set.

Also, the formatted I/O buffers of a session to a given device are reset whenever that device is cleared. Invoking the `viClear()` operation will flush the read buffer and discard the contents of the write buffers.

Related Items

See the `viSetBuf()` description in this chapter. Also see the *INTFC Resource*, *INSTR Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viGetAttribute

Purpose

Retrieves the state of an attribute.

C Syntax

```
ViStatus viGetAttribute(ViObject vi, ViAttr attribute,
void * attrState)
```

Visual Basic Syntax

```
viGetAttribute&(ByVal vi&, ByVal attribute&, attrState as Any)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session, event, or find list.
attribute	IN	Resource attribute for which the state query is made.
attrState	OUT	The state of the queried attribute for a specified resource. The interpretation of the returned value is defined by the individual object.

Return Values

Completion Codes	Description
VI_SUCCESS	Attribute retrieved successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given object reference is invalid.
VI_ERROR_NSUP_ATTR	The specified attribute is not defined by the referenced object.

Description

The `viGetAttribute()` operation is used to retrieve the state of an attribute for the specified session, event, or find list.

The output parameter **attrState** is of the type of the attribute actually being retrieved. For example, when retrieving an attribute that is defined as a `ViBoolean`, your application should pass a reference to a variable of type `ViBoolean`. Similarly, if the attribute is defined as being `ViUInt32`, your application should pass a reference to a variable of type `ViUInt32`.

Related Items

See the `viSetAttribute()` description in this chapter. Also see the *VISA Resource Template* description in Appendix B, *Resources*, and the attribute descriptions in Chapter 3, *Attributes*.

viGpibCommand

Purpose

Write GPIB command bytes on the bus.

C Syntax

```
ViStatus viGpibCommand (ViSession vi, ViBuf buf, ViUInt32 count,
ViPUInt32 retCount)
```

Visual Basic Syntax

```
viGpibCommand&(ByVal vi&, ByVal buf$, ByVal count&, retCount&)
```

Resource Classes

GPIB INTFC

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	IN	Buffer containing valid GPIB commands.
count	IN	Number of bytes to be written.
retCount	OUT	Number of bytes actually transferred.

Return Values

Completion Code	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.

Error Codes	Description
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_INV_SETUP	Unable to start write operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_NCIC	The interface associated with the given **vi** is not currently the controller in charge.
VI_ERROR_NLISTENERS	No Listeners condition is detected (both NRFD and NDAC are deasserted).
VI_ERROR_IO	An unknown I/O error occurred during transfer.

Description

This operation attempts to write **count** number of bytes of GPIB commands to the interface bus specified by **vi**. This operation is valid only on GPIB INTFC (interface) sessions. This operation returns only when the transfer terminates.

If you pass VI_NULL as the **retCount** parameter to the viGpibCommand() operation, the number of bytes transferred will not be returned. This may be useful if it is important to know only whether the operation succeeded or failed. The command bytes contained in **buf** should be valid IEEE 488-defined Multiline Interface Messages.

Related Items

See the *INTFC Resource* description in Appendix B, *Resources*.

viGpibControlATN

Purpose

Controls the state of the GPIB ATN interface line, and optionally the active controller state of the local interface.

C Syntax

```
ViStatus viGpibControlATN(ViSession vi, ViUInt16 mode)
```

Visual Basic Syntax

```
viGpibControlATN& (ByVal vi&, ByVal mode%)
```

Resource Classes

GPIB INTFC

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
mode	IN	Specifies the state of the ATN line and optionally the local active controller state. See the *Description* section for actual values.

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.

Error Codes	Description
VI_ERROR_NCIC	The interface associated with this session is not currently the controller in charge.
VI_ERROR_INV_MODE	The value specified by the **mode** parameter is invalid.
VI_ERROR_NSUP_MODE	The specified **mode** is not supported by this VISA implementation.

Description

This operation asserts or deasserts the GPIB ATN interface line according to the specified mode. The **mode** can also specify whether the local interface should acquire or release Controller Active status. This operation is valid only on GPIB INTFC (interface) sessions. The following table lists valid values for the **mode** parameter.

Mode	Action Description
VI_GPIB_ATN_DEASSERT	Deassert ATN line.
VI_GPIB_ATN_ASSERT	Assert ATN line synchronously (in 488 terminology). If a data handshake is in progress, ATN will not be asserted until the handshake is complete.
VI_GPIB_ATN_DEASSERT_HANDSHAKE	Deassert ATN line, and enter shadow handshake mode. The local board will participate in data handshakes as an Acceptor without actually reading the data.
VI_GPIB_ATN_ASSERT_IMMEDIATE	Assert ATN line asynchronously (in 488 terminology). This should generally be used only under error conditions.

It is generally not necessary to use the `viGpibControlATN()` operation in most applications. Other operations such as `viGpibCommand()` and `viGpibPassControl()` modify the ATN and/or CIC state automatically.

Related Items

See the `viGpibControlREN()` description in this chapter. Also see the *INTFC Resource* description in Appendix B, *Resources*.

viGpibControlREN

Purpose

Controls the state of the GPIB Remote Enable (REN) interface line, and optionally the remote/local state of the device.

C Syntax

```
ViStatus viGpibControlREN(ViSession vi, ViUInt16 mode)
```

Visual Basic Syntax

```
viGpibControlREN&(ByVal vi&, ByVal mode%)
```

Resource Classes

GPIB INSTR, GPIB INTFC

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
mode	IN	Specifies the state of the REN line and optionally the device remote/local state. See the *Description* section for actual values.

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.

Error Codes	Description
VI_ERROR_NCIC	The interface associated with this session is not currently the controller in charge.
VI_ERROR_NLISTENERS	No-listeners condition is detected (both NRFD and NDAC are unasserted).
VI_ERROR_NSYS_CNTLR	The interface associated with this session is not the system controller.
VI_ERROR_INV_MODE	The value specified by the **mode** parameter is invalid.

Description

The viGpibControlREN() operation asserts or unasserts the GPIB REN interface line according to the specified mode. The mode can also specify whether the device associated with this session should be placed in locate state (before deasserting REN) or remote state (after asserting REN). This operation is valid only if the GPIB interface associated with the session specified by **vi** is currently the system controller.

The following table lists special values for the **mode** parameter.

Value	Description
VI_GPIB_REN_DEASSERT	Deassert REN line.
VI_GPIB_REN_ASSERT	Assert REN line.
VI_GPIB_REN_DEASSERT_GTL	Send the Go To Local (GTL) command and deassert REN line.
VI_GPIB_REN_ASSERT_ADDRESS	Assert REN line and address device.
VI_GPIB_REN_ASSERT_LLO	Send LLO to any devices that are addressed to listen.
VI_GPIB_REN_ASSERT_ADDRESS_LLO	Address this device and send it LLO, putting it in RWLS.
VI_GPIB_REN_ADDRESS_GTL	Send the Go To Local command (GTL) to this device.

Related Items

See the viGpibControlATN() description in this chapter. Also see the *INSTR Resource* and *INTFC Resource* descriptions in Appendix B, *Resources*.

viGpibPassControl

Purpose

Tell the GPIB device at the specified address to become controller in charge (CIC).

C Syntax

```
ViStatus viGpibPassControl(ViSession vi, ViUInt16 primAddr,
ViUInt16 secAddr)
```

Visual Basic Syntax

```
viGPIBPassControl& (ByVal vi&, ByVal primAddr%, ByValsec Addr%)
```

Resource Classes

GPIB INTFC

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
primAddr	IN	Primary address of the GPIB device to which you want to pass control.
secAddr	IN	Secondary address of the targeted GPIB device. If the targeted device does not have a secondary address, this parameter should contain the value VI_NO_SEC_ADDR.

Return Values

Completion Code	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.

Error Codes	Description
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_NCIC	The interface associated with the given **vi** is not currently the controller in charge.
VI_ERROR_NLISTENERS	No Listeners condition is detected (both NRFD and NDAC are deasserted).
VI_ERROR_IO	An unknown I/O error occurred during transfer.

Description

This operation passes controller in charge status to the device indicated by **primAddr** and **secAddr**, and then deasserts the ATN line. This operation assumes that the targeted device has controller capability. This operation is valid only on GPIB INTFC (interface) sessions.

Related Items

See the *INTFC Resource* description in Appendix B, *Resources*.

viGpibSendIFC

Purpose

Pulse the interface clear line (IFC) for at least 100 microseconds.

C Syntax

```
ViStatus viGpibSendIFC (ViSession vi)
```

Visual Basic Syntax

```
viGpibSendIFC& (ByVal vi&)
```

Resource Classes

GPIB INTFC

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_NSYS_CNTLR	The interface associated with this session is not the system controller.

Description

This operation asserts the IFC line and becomes controller in charge (CIC). The local board must be the system controller. This operation is valid only on GPIB INTFC (interface) sessions.

Related Items

See the *INTFC Resource* description in Appendix B, *Resources*.

viIn8/viIn16/viIn32

Purpose

Reads in an 8-bit, 16-bit, or 32-bit value from the specified memory space and offset.

C Syntax

```
ViStatus viIn8(ViSession vi, ViUInt16 space, ViBusAddress offset,
ViPUInt8 val8)

ViStatus viIn16(ViSession vi, ViUInt16 space, ViBusAddress offset,
ViPUInt16 val16)

ViStatus viIn32(ViSession vi, ViUInt16 space, ViBusAddress offset,
ViPUInt32 val32)
```

Visual Basic Syntax

```
viIn8&(ByVal vi&, ByVal space%, ByVal offset&, val8 as Byte)

viIn16&(ByVal vi&, ByVal space%, ByVal offset&, val16%)

viIn32&(ByVal vi&, ByVal space%, ByVal offset&, val32&)
```

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, PXI INSTR, VXI INSTR, VXI MEMACC

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
space	IN	Specifies the address space. Refer to the table included in the *Description* section for more information.
offset	IN	Offset (in bytes) of the address or register from which to read.
val8, **val16**, or **val32**	OUT	Data read from bus (8 bits for `viIn8()`, 16 bits for `viIn16()`, and 32 bits for `viIn32()`).

Return Values

Completion Codes	Description
`VI_SUCCESS`	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SPACE	Invalid address **space** specified.
VI_ERROR_INV_OFFSET	Invalid **offset** specified.
VI_ERROR_NSUP_OFFSET	Specified **offset** is not accessible from this hardware.
VI_ERROR_NSUP_WIDTH	Specified width is not supported by this hardware.
VI_ERROR_NSUP_ALIGN_OFFSET	The specified **offset** is not properly aligned for the access width of the operation.
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).

Description

The `viIn`*`XX`*`()` operations use the specified address **space** to read in 8, 16, or 32 bits of data, respectively, from the specified **offset**. These operations do not require `viMapAddress()` to be called prior to their invocation.

The following table lists the valid entries for specifying address **space**.

Value	Description

VXI, VME, and GPIB-VXI	`VI_A16_SPACE (1)`
	`VI_A24_SPACE (2)`
	`VI_A32_SPACE (3)`

PXI	`VI_PXI_CFG_SPACE (10)` `VI_PXI_BAR0_SPACE (11)` to `VI_PXI_BAR5_SPACE (16)`

INSTR Specific

Notice that **offset** specified in the `viIn8()`, `viIn16()`, and `viIn32()` operations for an INSTR Resource is the offset address relative to the device's allocated address base for the corresponding address space that was specified. For example, if **space** specifies `VI_A16_SPACE`, then **offset** specifies the offset from the logical address base address of the specified VXI device. If **space** specifies `VI_A24_SPACE` or `VI_A32_SPACE`, then **offset** specifies the offset from the base address of the VXI device's memory space allocated by the VXI Resource Manager within VXI A24 or A32 space.

MEMACC Specific

For a MEMACC Resource, the **offset** parameter specifies an absolute address.

Related Items

See the `viOut8/viOut16/viOut32()` descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

viInstallHandler

Purpose

Installs handlers for event callbacks.

C Syntax

```
ViStatus viInstallHandler(ViSession vi, ViEventType eventType,
ViHndlr handler, ViAddr userHandle)
```

Visual Basic Syntax

N/A

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
eventType	IN	Logical event identifier.
handler	IN	Interpreted as a valid reference to a handler to be installed by a client application.
userHandle	IN	A value specified by an application that can be used for identifying handlers uniquely for an event type.

Return Values

Completion Codes	Description
VI_SUCCESS	Event handler installed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_INV_EVENT	Specified **eventType** is not supported by the resource.
VI_ERROR_INV_HNDLR_REF	The given **handler** reference is invalid.
VI_ERROR_HNDLR_NINSTALLED	The handler was not installed. This may be returned if an application attempts to install multiple handlers for the same event on the same session.

Description

The `viInstallHandler()` operation allows applications to install handlers on sessions. The handler specified in the **handler** parameter is installed along with any previously installed handlers for the specified event. Applications can specify a value in the **userHandle** parameter that is passed to the handler on its invocation. VISA identifies handlers uniquely using the handler reference and this value.

VISA allows applications to install multiple handlers for an **eventType** on the same session. You can install multiple handlers through multiple invocations of the `viInstallHandler()` operation, where each invocation adds to the previous list of handlers. If more than one handler is installed for an **eventType**, each of the handlers is invoked on every occurrence of the specified event(s). VISA specifies that the handlers are invoked in Last In First Out (LIFO) order.

Related Items

See the `viEventHandler()`, `viEnableEvent()`, and `viUninstallHandler()` descriptions. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

viLock

Purpose

Establishes an access mode to the specified resource.

C Syntax

```
ViStatus viLock(ViSession vi, ViAccessMode lockType,
ViUInt32 timeout, ViKeyId requestedKey, ViChar accesskey[])
```

Visual Basic Syntax

```
viLock&(ByVal vi&, ByVal lockType&, ByVal timeout&,
ByVal requestedKey$, ByVal accesskey$)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
lockType	IN	Specifies the type of lock requested, either `VI_EXCLUSIVE_LOCK` (1) or `VI_SHARED_LOCK` (2).
timeout	IN	Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error.
requestedKey	IN	This parameter is not used and should be set to `VI_NULL` when **lockType** is `VI_EXCLUSIVE_LOCK`. See the *Description* section for more details about using `VI_SHARED_LOCK`.
accessKey	OUT	This parameter should be set to `VI_NULL` when **lockType** is `VI_EXCLUSIVE_LOCK`. When **lockType** is `VI_SHARED_LOCK`, the resource returns a unique access key for the lock if the operation succeeds. This **accessKey** can then be passed to other sessions to share the lock.

Return Values

Completion Codes	Description
VI_SUCCESS	Specified access mode was acquired.
VI_SUCCESS_NESTED_EXCLUSIVE	Specified access mode is successfully acquired, and this session has nested exclusive locks.
VI_SUCCESS_NESTED_SHARED	Specified access mode is successfully acquired, and this session has nested shared locks.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified **lockType** cannot be obtained because the resource is already locked with a lock type incompatible with the lock requested.
VI_ERROR_INV_LOCK_TYPE	Specified **lockType** is not supported by this resource.
VI_ERROR_INV_ACCESS_KEY	The **requestedKey** value passed in is not a valid **accessKey** to the specified resource.
VI_ERROR_TMO	Specified **lockType** could not be obtained within the specified **timeout** period.

Description

This operation is used to obtain a lock on the specified resource. The caller can specify the type of lock requested—exclusive or shared lock—and the length of time the operation will suspend while waiting to acquire the lock before timing out. This operation can also be used for sharing and nesting locks.

The **requestedKey** and the **accessKey** parameters apply only to shared locks. These parameters are not applicable when using the lock type VI_EXCLUSIVE_LOCK; in this case, **requestedKey** and **accessKey** should be set to VI_NULL. VISA allows user applications to specify a key to be used for lock sharing, through the use of the **requestedKey** parameter. Alternatively, a user application can pass VI_NULL for the **requestedKey** parameter when obtaining a shared lock, in which case VISA will generate a unique access key and return it through the **accessKey** parameter. If a user application does specify a **requestedKey** value, VISA will try to use this value for the **accessKey**. As long as the resource is not locked, VISA

will use the **requestedKey** as the access key and grant the lock. When the operation succeeds, the **requestedKey** will be copied into the user buffer referred to by the **accessKey** parameter.

Note If requesting a `VI_SHARED_LOCK`, the size of the **accessKey** parameter should be at least 256 bytes.

The session that gained a shared lock can pass the **accessKey** to other sessions for the purpose of sharing the lock. The session wanting to join the group of sessions sharing the lock can use the key as an input value to the **requestedKey** parameter. VISA will add the session to the list of sessions sharing the lock, as long as the **requestedKey** value matches the **accessKey** value for the particular resource. The session obtaining a shared lock in this manner will then have the same access privileges as the original session that obtained the lock.

It is also possible to obtain nested locks through this operation. To acquire nested locks, invoke the `viLock()` operation with the same lock type as the previous invocation of this operation. For each session, `viLock()` and `viUnlock()` share a lock count, which is initialized to 0. Each invocation of `viLock()` for the same session (and for the same **lockType**) increases the lock count. In the case of a shared lock, it returns with the same **accessKey** every time. When a session locks the resource a multiple number of times, it is necessary to invoke the `viUnlock()` operation an equal number of times in order to unlock the resource. That is, the lock count increments for each invocation of `viLock()`, and decrements for each invocation of `viUnlock()`. A resource is actually unlocked only when the lock count is 0.

The VISA locking mechanism enforces arbitration of accesses to resources on an individual basis. If a session locks a resource, operations invoked by other sessions to the same resource are serviced or returned with a locking error, depending on the operation and the type of lock used. If a session has an exclusive lock, other sessions cannot modify global attributes or invoke operations, but can still get attributes and set local attributes. If the session has a shared lock, other sessions that have shared locks can also modify global attributes and invoke operations. Regardless of which type of lock a session has, if the session is closed without first being unlocked, VISA automatically performs a viUnlock() on that session.

The locking mechanism works for all processes and resources existing on the same computer. When using remote resources, however, the networking protocol may not provide the ability to pass lock requests to the remote device or resource. In this case, locks will behave as expected from multiple sessions on the same computer, but not necessarily on the remote device. For example, when using the VXI-11 protocol, exclusive lock requests can be sent to a device, but shared locks can only be handled locally.

Related Items

See the `viUnlock()` description in this chapter. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

viMapAddress

Purpose

Maps the specified memory space into the process's address space.

C Syntax

```
ViStatus viMapAddress(ViSession vi, ViUInt16 mapSpace,
ViBusAddress mapBase, ViBusSize mapSize, ViBoolean access,
ViAddr suggested, ViPAddr address)
```

Visual Basic Syntax

```
viMapAddress&(ByVal vi&, ByVal mapSpace%, ByVal mapBase&,
ByVal mapSize&, ByVal access%, ByVal suggested&, address&)
```

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, PXI INSTR, VXI INSTR, VXI MEMACC

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
mapSpace	IN	Specifies the address space to map. Refer to the *Description* section for more information.
mapBase	IN	Offset (in bytes) of the memory to be mapped. Refer to the *Description* section for more information.
mapSize	IN	Amount of memory to map (in bytes).
access	IN	`VI_FALSE` (0).
suggested	IN	If **suggested** parameter is not `VI_NULL` (0), the operating system attempts to map the memory to the address specified in **suggested**. There is no guarantee, however, that the memory will be mapped to that address. This operation may map the memory into an address region different from **suggested**.
address	OUT	Address in your process space where the memory was mapped.

Return Values

Completion Codes	Description
VI_SUCCESS	Mapping successful.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_INV_SPACE	Invalid address space specified.
VI_ERROR_INV_OFFSET	Invalid offset specified.
VI_ERROR_NSUP_OFFSET	Specified region is not accessible from this hardware.
VI_ERROR_TMO	`viMapAddress()` could not acquire resource or perform mapping before the timer expired.
VI_ERROR_INV_SIZE	Invalid size of window specified.
VI_ERROR_ALLOC	Unable to allocate window of at least the requested size.
VI_ERROR_INV_ACC_MODE	Invalid access mode.
VI_ERROR_WINDOW_MAPPED	The specified session already contains a mapped window.
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).

Description

The `viMapAddress()` operation maps in a specified memory space. The memory space that is mapped is dependent on the type of interface specified by the **vi** parameter and the **mapSpace** parameter. The **address** parameter returns the address in your process space where memory is mapped. The following table lists the valid entries for the **mapSpace** parameter.

Value	Description

VXI, VME, and GPIB-VXI	`VI_A16_SPACE (1)`
	`VI_A24_SPACE (2)`
	`VI_A32_SPACE (3)`

PXI	`VI_PXI_CFG_SPACE (10)` `VI_PXI_BAR0_SPACE (11)` to `VI_PXI_BAR5_SPACE (16)`

INSTR Specific

Notice that **mapBase** specified in the `viMapAddress()` operation for an INSTR Resource is the offset address relative to the device's allocated address base for the corresponding address space that was specified. For example, if **mapSpace** specifies `VI_A16_SPACE`, then **mapBase** specifies the offset from the logical address base address of the specified VXI device. If **mapSpace** specifies `VI_A24_SPACE` or `VI_A32_SPACE`, then **mapBase** specifies the offset from the base address of the VXI device's memory space allocated by the VXI Resource Manager within VXI A24 or A32 space.

MEMACC Specific

For a MEMACC Resource, the **mapBase** parameter specifies an absolute address.

Related Items

See the `viUnmapAddress()` description in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

viMapTrigger

Purpose

Map the specified trigger source line to the specified destination line.

C Syntax

```
viStatus viMapTrigger(ViSession vi, ViInt16 trigSrc,
ViInt16 trigDest,ViUInt16 mode)
```

Visual Basic Syntax

```
viMapTrigger& (ByVal vi&, ByVal trigSrc%, ByVal trigDest%,
ByVal mode%)
```

Resource Classes

GPIB-VXI BACKPLANE, VXI BACKPLANE

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
trigSrc	IN	Source line from which to map. See the *Description* section for actual values.
trigDest	IN	Destination line to which to map. See the *Description* section for actual values.
mode	IN	VI_NULL

Return Values

Completion Code	Description
VI_SUCCESS	Operation completed successfully.
VI_SUCCESS_TRIG_MAPPED	The path from **trigSrc** to **trigDest** is already mapped.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_INV_MODE	The value specified by the **mode** parameter is invalid.
VI_ERROR_LINE_IN_USE	One of the specified lines (**trigSrc** or **trigDest**) is currently in use.
VI_ERROR_INV_LINE	One of the specified lines (**trigSrc** or **trigDest**) is invalid.
VI_ERROR_NSUP_LINE	One of the specified lines (**trigSrc** or **trigDest**) is not supported by this VISA implementation.

Description

This operation can be used to map one trigger line to another. This operation is valid only on BACKPLANE (mainframe) sessions.

Special Values for trigSrc and trigDest Parameters

Value	Action Description
VI_TRIG_TTL0 - VI_TRIG_TTL7	Map the specified VXI TTL trigger line.
VI_TRIG_ECL0 - VI_TRIG_ECL1	Map the specified VXI ECL trigger line.
VI_TRIG_PANEL_IN	Map the controller's front panel trigger input line.
VI_TRIG_PANEL_OUT	Map the controller's front panel trigger output line.

If this operation is called multiple times on the same BACKPLANE Resource with the same source trigger line and different destination trigger lines, the result will be that when the source trigger line is asserted, all of the specified destination trigger lines will also be asserted. If this operation is called multiple times on the same BACKPLANE Resource with different

source trigger lines and the same destination trigger line, the result will be that when any of the specified source trigger lines is asserted, the destination trigger line will also be asserted.

Note Mapping a trigger line (as either source or destination) multiple times requires special hardware capabilities and is not guaranteed to be implemented.

Related Items

See the *BACKPLANE Resource* description in Appendix B, *Resources*.

viMemAlloc

Purpose

Allocates memory from a device's memory region.

C Syntax

```
ViStatus viMemAlloc(ViSession vi, ViBusSize size,
ViPBusAddress offset)
```

Visual Basic Syntax

```
viMemAlloc&(ByVal vi&, ByVal size&, offset&)
```

Resource Classes

GPIB-VXI INSTR, VXI INSTR

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
size	IN	Specifies the size of the allocation.
offset	OUT	Returns the offset of the allocated device memory.

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_INV_SIZE	Invalid **size** specified.

Error Codes	Description
VI_ERROR_ALLOC	Unable to allocate shared memory block of the requested **size**.
VI_ERROR_MEM_NSHARED	The device does not export any memory.

Description

The viMemAlloc() operation returns an offset into a device's memory region that has been allocated for use by this session. If the device to which the given **vi** refers is located on the local interface card, the memory can be allocated either on the device itself or on the computer's system memory.

The memory region referenced by the **offset** that is returned from this operation can be accessed with the high-level operations viMoveIn*XX*() and viMoveOut*XX*(), or it can be mapped using viMapAddress().

Related Items

See the viMapAddress(), viMemFree(), viMoveIn8/viMoveIn16/viMoveIn32(), and viMoveOut8/viMoveOut16/viMoveOut32() descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

viMemFree

Purpose

Frees memory previously allocated using the `viMemAlloc()` operation.

C Syntax

```
ViStatus viMemFree(ViSession vi, ViBusAddress offset)
```

Visual Basic Syntax

```
viMemFree&(ByVal vi&, ByVal offset&)
```

Resource Classes

GPIB-VXI INSTR, VXI INSTR

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
offset	IN	Specifies the memory previously allocated with `viMemAlloc()`.

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_INV_OFFSET	Invalid **offset** specified.
VI_ERROR_WINDOW_MAPPED	The specified **offset** is currently in use by `viMapAddress()`.

Description

The `viMemFree()` operation frees the memory previously allocated using `viMemAlloc()`. If the specified **offset** has been mapped using `viMapAddress()`, it must be unmapped before it can be freed.

Related Items

See the `viMapAddress()`, `viMemAlloc()`, and `viUnmapAddress()` descriptions in this chapter. Also see the *INSTR Resource* description in Appendix B, *Resources*.

viMove

Purpose

Moves a block of data.

C Syntax

```
ViStatus viMove(ViSession vi, ViUInt16 srcSpace,
ViBusAddress srcOffset, ViUInt16 srcWidth, ViUInt16 destSpace,
ViBusAddress destOffset, ViUInt16 destWidth, ViBusSize length)
```

Visual Basic Syntax

```
viMove&(ByVal vi&, ByVal srcSpace%, ByVal srcOffset&, ByVal srcWidth%,
ByVal destSpace%, ByVal destOffset&, ByVal destWidth%, ByVal length&)
```

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, PXI INSTR, VXI INSTR, VXI MEMACC

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
srcSpace	IN	Specifies the address space of the source.
srcOffset	IN	Offset of the starting address or register from which to read.
srcWidth	IN	Specifies the data width of the source.
destSpace	IN	Specifies the address space of the destination.
destOffset	IN	Offset of the starting address or register to which to write.
destWidth	IN	Specifies the data width of the destination.
length	IN	Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SPACE	Invalid source or **destSpace** specified.
VI_ERROR_INV_OFFSET	Invalid source or **destWidth** specified.
VI_ERROR_INV_WIDTH	Invalid source or **destWidth** specified.
VI_ERROR_NSUP_OFFSET	Specified source or destination offset is not accessible from this hardware.
VI_ERROR_NSUP_VAR_WIDTH	Cannot support source and destination widths that are different.
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_NSUP_WIDTH	Specified width is not supported by this hardware.
VI_ERROR_NSUP_ALIGN_OFFSET	The specified offset is not properly aligned for the access width of the operation.
VI_ERROR_INV_LENGTH	Invalid length specified.

Description

The `viMove()` operation moves data from the specified source to the specified destination. The source and the destination can either be local memory or the offset of the interface with which this MEMACC Resource is associated. This operation uses the specified data width and address space. In some systems, such as VXI, users can specify additional settings for the transfer, such as byte order and access privilege, by manipulating the appropriate attributes.

The following table lists the valid entries for specifying address space.

Value	Description
VI_A16_SPACE (1)	Address the A16 address space of the VXI/MXI bus.
VI_A24_SPACE (2)	Address the A24 address space of the VXI/MXI bus.
VI_A32_SPACE (3)	Address the A32 address space of the VXI/MXI bus.
VI_LOCAL_SPACE (0)	Address process-local memory (using a virtual address).

The following table lists the valid entries for specifying widths.

Value	Description
VI_WIDTH_8 (1)	Performs 8-bit (D08) transfers.
VI_WIDTH_16 (2)	Performs 16-bit (D16) transfers.
VI_WIDTH_32 (4)	Performs 32-bit (D32) transfers.

INSTR Specific

If **srcSpace** is not VI_LOCAL_SPACE, then **srcOffset** is a relative address of the device associated with the given INSTR Resource. Similarly, if **destSpace** is not VI_LOCAL_SPACE, then **destOffset** is a relative address of the device associated with the given INSTR Resource.

The primary intended use of this operation with an INSTR session is to synchronously move data to or from the device. Therefore, either the **srcSpace** or **destSpace** parameter will usually be VI_LOCAL_SPACE.

MEMACC Specific

The **destOffset** and **srcOffset** parameters specify absolute addresses. Notice also that the **length** specified in the viMove() operation is the number of elements (of the size corresponding to the **srcWidth** parameter) to transfer, beginning at the specified offsets. Therefore, **srcOffset + length*srcWidth** cannot exceed the total amount of memory exported by the given **srcSpace**. Similarly, **destOffset** + **length*srcWidth** cannot exceed the total amount of memory exported by the given **destSpace**.

Related Items

See the viMoveAsync() description in this chapter. See the VI_ATTR_DEST_INCREMENT and VI_ATTR_SRC_INCREMENT descriptions in Chapter 3, *Attributes*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

viMoveAsync

Purpose

Moves a block of data asynchronously.

C Syntax

```
ViStatus viMoveAsync(ViSession vi, ViUInt16 srcSpace,
ViBusAddress srcOffset, ViUInt16 srcWidth, ViUInt16 destSpace,
ViBusAddress destOffset, ViUInt16 destWidth, ViBusSize length,
ViPJobId jobId)
```

Visual Basic Syntax

N/A

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, PXI INSTR, VXI INSTR, VXI MEMACC

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
srcSpace	IN	Specifies the address space of the source.
srcOffset	IN	Offset of the starting address or register from which to read.
srcWidth	IN	Specifies the data width of the source.
destSpace	IN	Specifies the address space of the destination.
destOffset	IN	Offset of the starting address or register to which to write.
destWidth	IN	Specifies the data width of the destination.
length	IN	Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
jobId	OUT	Job identifier of this asynchronous move operation.

Return Values

Completion Codes	Description
VI_SUCCESS	Asynchronous operation successfully queued.
VI_SUCCESS_SYNC	Operation performed synchronously.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_QUEUE_ERROR	Unable to queue move operation.
VI_ERROR_IN_PROGRESS	Unable to queue the asynchronous operation because there is already an operation in progress.

Description

The viMoveAsync() operation asynchronously moves data from the specified source to the specified destination. This operation queues up the transfer in the system, then it returns immediately without waiting for the transfer to carry out or complete. When the transfer terminates, a VI_EVENT_IO_COMPLETION event is generated, which indicates the status of the transfer.

This operation returns **jobId**, which you can use either with viTerminate() to abort the operation or with VI_EVENT_IO_COMPLETION events to identify which asynchronous move operations completed. VISA will never return VI_NULL for a valid **jobId**.

The source and the destination can either be local memory or the offset of the interface with which this INSTR or MEMACC Resource is associated. This operation uses the specified data width and address space. In some systems, such as VXI, users can specify additional settings for the transfer, such as byte order and access privilege, by manipulating the appropriate attributes.

The following table lists the valid entries for specifying address space.

Value	Description
VI_A16_SPACE (1)	Address the A16 address space of the VXI/MXI bus.
VI_A24_SPACE (2)	Address the A24 address space of the VXI/MXI bus.
VI_A32_SPACE (3)	Address the A32 address space of the VXI/MXI bus.
VI_LOCAL_SPACE (0)	Address process-local memory (using a virtual address).

The following table lists the valid entries for specifying widths.

Value	Description
VI_WIDTH_8 (1)	Performs 8-bit (D08) transfers.
VI_WIDTH_16 (2)	Performs 16-bit (D16) transfers.
VI_WIDTH_32 (4)	Performs 32-bit (D32) transfers.

INSTR Specific

If **srcSpace** is not VI_LOCAL_SPACE, then **srcOffset** is a relative address of the device associated with the given INSTR Resource. Similarly, if **destSpace** is not VI_LOCAL_SPACE, then **destOffset** is a relative address of the device associated with the given INSTR Resource.

The primary intended use of this operation with an INSTR session is to asynchronously move data to or from the device. Therefore, either the **srcSpace** or **destSpace** parameter will usually be VI_LOCAL_SPACE.

MEMACC Specific

The **destOffset** and **srcOffset** parameters specify absolute addresses. Notice also that the **length** specified in the viMoveAsync() operation is the number of elements (of the size corresponding to the **srcWidth** parameter) to transfer, beginning at the specified offsets. Therefore, **srcOffset + length*srcWidth** cannot exceed the total amount of memory exported by the given **srcSpace**. Similarly, **destOffset** + **length*srcWidth** cannot exceed the total amount of memory exported by the given **destSpace**.

Related Items

See the viMove() description in this chapter. Also see the VI_ATTR_DEST_INCREMENT and VI_ATTR_SRC_INCREMENT descriptions in Chapter 3, *Attributes*. See the VI_EVENT_IO_COMPLETION description in Chapter 4, *Events*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

viMoveIn8/viMoveIn16/viMoveIn32

Purpose

Moves a block of data from the specified address space and offset to local memory.

C Syntax

```
ViStatus viMoveIn8(ViSession vi, ViUInt16 space, ViBusAddress offset,
ViBusSize length, ViAUInt8 buf8)

ViStatus viMoveIn16(ViSession vi, ViUInt16 space,
ViBusAddress offset, ViBusSize length, ViAUInt16 buf16)

ViStatus viMoveIn32(ViSession vi, ViUInt16 space,
ViBusAddress offset, ViBusSize length, ViAUInt32 buf32)
```

Visual Basic Syntax

```
viMoveIn8&(ByVal vi&, ByVal space%, ByVal offset&, ByVal length&,
buf8 as Byte)

viMoveIn16&(ByVal vi&, ByVal space%, ByVal offset&, ByVal length&,
buf16%)

viMoveIn32&(ByVal vi&, ByVal space%, ByVal offset&, ByVal length&,
buf32&)
```

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, PXI INSTR, VXI INSTR, VXI MEMACC

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
space	IN	Specifies the address space. Refer to the table included in the *Description* section.
offset	IN	Offset (in bytes) of the starting address to read.
length	IN	Number of elements to transfer, where the data width of the elements to transfer is identical to data width (8, 16, or 32 bits).
buf8, **buf16**, or **buf32**	OUT	Data read from bus (8 bits for `viMoveIn8()`, 16 bits for `viMoveIn16()`, and 32 bits for `viMoveIn32()`).

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SPACE	Invalid address space specified.
VI_ERROR_INV_OFFSET	Invalid offset specified.
VI_ERROR_NSUP_OFFSET	Specified offset is not accessible from this hardware.
VI_ERROR_NSUP_WIDTH	Specified width is not supported by this hardware.
VI_ERROR_INV_LENGTH	Invalid length specified.
VI_ERROR_NSUP_ALIGN_OFFSET	The specified offset is not properly aligned for the access width of the operation.
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).

Description

The viMoveIn*XX*() operations use the specified address space to read in 8, 16, or 32 bits of data, respectively, from the specified offset. These operations do not require viMapAddress() to be called prior to their invocation.

The following table lists the valid entries for specifying address space.

Value	Description
VXI, VME, and GPIB-VXI	VI_A16_SPACE (1) VI_A24_SPACE (2) VI_A32_SPACE (3)
PXI	VI_PXI_CFG_SPACE (10) VI_PXI_BAR0_SPACE (11) to VI_PXI_BAR5_SPACE (16)

INSTR Specific

Notice that **offset** specified in the viMoveIn8(), viMoveIn16(), and viMoveIn32() operations for an INSTR Resource is the offset address relative to the device's allocated address base for the corresponding address space that was specified. For example, if **space** specifies VI_A16_SPACE, then **offset** specifies the offset from the logical address base address of the specified VXI device. If **space** specifies VI_A24_SPACE or VI_A32_SPACE, then **offset** specifies the offset from the base address of the VXI device's memory space allocated by the VXI Resource Manager within VXI A24 or A32 space.

Notice also that the **length** specified in the viMoveIn*XX*() operations for an INSTR Resource is the number of elements (of the **size** corresponding to the operation) to transfer, beginning at the specified **offset**. Therefore, **offset + length*size** cannot exceed the amount of memory exported by the device in the given **space**.

MEMACC Specific

For a MEMACC Resource, the **offset** parameter specifies an absolute address.

Notice also that the **length** specified in the viMoveIn*XX*() operations for a MEMACC Resource is the number of elements (of the **size** corresponding to the operation) to transfer, beginning at the specified **offset**. Therefore, **offset + length*size** cannot exceed the total amount of memory available in the given **space**.

Related Items

See the viMoveOut8/viMoveOut16/viMoveOut32() descriptions in this chapter. See the VI_ATTR_DEST_INCREMENT description in Chapter 3, *Attributes*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

viMoveOut8/viMoveOut16/viMoveOut32

Purpose

Moves a block of data from local memory to the specified address space and offset.

C Syntax

```
ViStatus viMoveOut8(ViSession vi, ViUInt16 space,
ViBusAddress offset, ViBusSize length, ViAUInt8 buf8)

ViStatus viMoveOut16(ViSession vi, ViUInt16 space,
ViBusAddress offset, ViBusSize length, ViAUInt16 buf16)

ViStatus viMoveOut32(ViSession vi, ViUInt16 space,
ViBusAddress offset, ViBusSize length, ViAUInt32 buf32)
```

Visual Basic Syntax

```
viMoveOut8&(ByVal vi&, ByVal space%, ByVal offset&,
ByVal length&,buf8 as Byte)

viMoveOut16&(ByVal vi&, ByVal space%, ByVal offset&,
ByVal length&, buf16%)

viMoveOut32&(ByVal vi&, ByVal space%, ByVal offset&,
ByVal length&, buf32&)
```

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, PXI INSTR, VXI INSTR, VXI MEMACC

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
space	IN	Specifies the address space. Refer to the table included in the *Description* section.
offset	IN	Offset (in bytes) of the device to write to.
length	IN	Number of elements to transfer, where the data width of the elements to transfer is identical to data width (8, 16, or 32 bits).
buf8, **buf16**, or **buf32**	IN	Data to write bus (8 bits for `viMoveOut8()`, 16 bits for `viMoveOut16()`, and 32 bits for `viMoveOut32()`).

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SPACE	Invalid address space specified.
VI_ERROR_INV_OFFSET	Invalid offset specified.
VI_ERROR_NSUP_OFFSET	Specified offset is not accessible from this hardware.
VI_ERROR_NSUP_WIDTH	Specified width is not supported by this hardware.
VI_ERROR_INV_LENGTH	Invalid length specified.
VI_ERROR_NSUP_ALIGN_OFFSET	The specified offset is not properly aligned for the access width of the operation.
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).

Description

The viMoveOut*XX*() operations use the specified address space to write 8, 16, or 32 bits of data, respectively, to the specified offset. These operations do not require viMapAddress() to be called prior to their invocation.

The following table lists the valid entries for specifying address space.

Interface	Values
VXI, VME, and GPIB-VXI	VI_A16_SPACE (1) VI_A24_SPACE (2) VI_A32_SPACE (3)
PXI	VI_PXI_CFG_SPACE (10) VI_PXI_BAR0_SPACE (11) to VI_PXI_BAR5_SPACE (16)

INSTR Specific

Notice that **offset** specified in the viMoveOut8(), viMoveOut16(), and viMoveOut32() operations for an INSTR Resource is the offset address relative to the device's allocated address base for the corresponding address space that was specified. For example, if **space** specifies VI_A16_SPACE, then **offset** specifies the offset from the logical address base address of the specified VXI device. If **space** specifies VI_A24_SPACE or VI_A32_SPACE, then **offset** specifies the offset from the base address of the VXI device's memory space allocated by the VXI Resource Manager within VXI A24 or A32 space.

Notice also that the **length** specified in the viMoveIn*XX*() operations for an INSTR Resource is the number of elements (of the **size** corresponding to the operation) to transfer, beginning at the specified **offset**. Therefore, **offset + length*size** cannot exceed the amount of memory exported by the device in the given **space**.

MEMACC Specific

For a MEMACC Resource, the **offset** parameter specifies an absolute address.

Notice also that the **length** specified in the viMoveOut*XX*() operations for a MEMACC Resource is the number of elements (of the **size** corresponding to the operation) to transfer, beginning at the specified **offset**. Therefore, **offset + length*size** cannot exceed the total amount of memory available in the given **space**.

Related Items

See the viMoveIn8/viMoveIn16/viMoveIn32() descriptions in this chapter. See the VI_ATTR_DEST_INCREMENT description in Chapter 3, *Attributes*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

viOpen

Purpose

Opens a session to the specified resource.

C Syntax

```
ViStatus viOpen(ViSession sesn, ViRsrc rsrcName,
ViAccessMode accessMode, ViUInt32 openTimeout, ViPSession vi)
```

Visual Basic Syntax

```
viOpen&(ByVal sesn&, ByVal rsrcName$, ByVal accessMode&,
ByVal openTimeout&, vi&)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
sesn	IN	Resource Manager session (should always be a session returned from `viOpenDefaultRM()`).
rsrcName	IN	Unique symbolic name of a resource. See the *Description* section for more information.
accessMode	IN	Specifies the mode by which the resource is to be accessed. See the *Description* section for valid values. If the parameter value is `VI_NULL`, the session uses VISA-supplied default values.
openTimeout	IN	Specifies the maximum time period (in milliseconds) that this operation waits before returning an error. This does not set the I/O timeout - to do that you must call `viSetAttribute()` with the attribute `VI_ATTR_TMO_VALUE`.
vi	OUT	Unique logical identifier reference to a session.

Return Values

Completion Codes	Description
VI_SUCCESS	Session opened successfully.
VI_SUCCESS_DEV_NPRESENT	Session opened successfully, but the device at the specified address is not responding.
VI_WARN_CONFIG_NLOADED	The specified configuration either does not exist or could not be loaded; using VISA-specified defaults.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **sesn** does not support this operation. This operation is supported only by a Resource Manager session.
VI_ERROR_INV_RSRC_NAME	Invalid resource reference specified. Parsing error.
VI_ERROR_INV_ACC_MODE	Invalid access mode.
VI_ERROR_RSRC_NFOUND	Insufficient location information or resource not present in the system.
VI_ERROR_ALLOC	Insufficient system resources to open a session.
VI_ERROR_RSRC_BUSY	The resource is valid, but VISA cannot currently access it.
VI_ERROR_RSRC_LOCKED	Specified type of lock cannot be obtained because the resource is already locked with a lock type incompatible with the lock requested.
VI_ERROR_TMO	A session to the resource could not be obtained within the specified openTimeout period.
VI_ERROR_LIBRARY_NFOUND	A code library required by VISA could not be located or loaded.
VI_ERROR_INTF_NUM_NCONFIG	The interface type is valid, but the specified interface number is not configured.

Description

The `viOpen()` operation opens a session to the specified resource. It returns a session identifier that can be used to call any other operations of that resource. The address string passed to `viOpen()` must uniquely identify a resource. The following table shows the grammar for the address string. Optional string segments are shown in square brackets ([ ]).

Interface	Syntax
VXI INSTR	VXI[*board*]::*VXI logical address*[::INSTR]
VXI MEMACC	VXI[*board*]::MEMACC
VXI BACKPLANE	VXI[*board*][::*mainframe logical address*] ::BACKPLANE
VXI SERVANT	VXI[*board*]::SERVANT
GPIB-VXI INSTR	GPIB-VXI[*board*]::*VXI logical address*[::INSTR]
GPIB-VXI MEMACC	GPIB-VXI[*board*]::MEMACC
GPIB-VXI BACKPLANE	GPIB-VXI[*board*][::*mainframe logical address*] ::BACKPLANE
GPIB INSTR	GPIB[*board*]::*primary address*[::*secondary address*][::INSTR]
GPIB INTFC	GPIB[*board*]::INTFC
GPIB SERVANT	GPIB[*board*]::SERVANT
PXI INSTR	PXI[*board*]::*device*[::*function*][::INSTR]
Serial INSTR	ASRL[*board*][::INSTR]
TCPIP INSTR	TCPIP[*board*]::*host address*[::*LAN device name*] [::INSTR]
TCPIP SOCKET	TCPIP[*board*]::*host address*::*port*::SOCKET

The VXI keyword is used for VXI instruments via either embedded or MXIbus controllers. The GPIB-VXI keyword is used for a GPIB-VXI controller. The GPIB keyword can be used to establish communication with a GPIB device. The ASRL keyword is used to establish communication with an asynchronous serial (such as RS-232) device.

The following table shows the default value for optional string segments.

Optional String Segments	Default Value
board	0
secondary address	none
LAN device name	inst0

The following table shows examples of address strings.

Address String	Description
VXI0::1::INSTR	A VXI device at logical address 1 in VXI interface VXI0.
GPIB-VXI::9::INSTR	A VXI device at logical address 9 in a GPIB-VXI controlled system.
GPIB::1::0::INSTR	A GPIB device at primary address 1 and secondary address 0 in GPIB interface 0.
ASRL1::INSTR	A serial device attached to interface ASRL1.
VXI::MEMACC	Board-level register access to the VXI interface.
GPIB-VXI1::MEMACC	Board-level register access to GPIB-VXI interface number 1.
GPIB2::INTFC	Interface or raw resource for GPIB interface 2.
VXI::1::BACKPLANE	Mainframe resource for chassis 1 on the default VXI system, which is interface 0.
GPIB-VXI2::BACKPLANE	Mainframe resource for default chassis on GPIB-VXI interface 2.
GPIB1::SERVANT	Servant/device-side resource for GPIB interface 1.
VXI0::SERVANT	Servant/device-side resource for VXI interface 0.
PXI::15::INSTR	PXI device number 15 on bus 0.
TCPIP0::1.2.3.4::999 ::SOCKET	Raw TCP/IP access to port 999 at the specified IP address.
TCPIP::dev@company.com ::INSTR	A TCP/IP device using VXI-11 located at the specified address. This uses the default LAN Device Name of inst0.

For the parameter **accessMode**, the value VI_EXCLUSIVE_LOCK (1) is used to acquire an exclusive lock immediately upon opening a session; if a lock cannot be acquired, the session is closed and an error is returned. The value VI_LOAD_CONFIG (4) is used to configure attributes to values specified by some external configuration utility. Multiple access modes can be used simultaneously by specifying a *bit-wise OR* of the values other than VI_NULL. NI-VISA currently supports VI_LOAD_CONFIG only on Serial INSTR sessions.

All resource strings returned by viFindRsrc() will always be recognized by viOpen(). However, viFindRsrc() will not necessarily return all strings that you can pass to viParseRsrc() or viOpen(). This is especially true for network and TCPIP resources.

Related Items

See the viClose(), viFindRsrc(), viOpenDefaultRM(), and viParseRsrc() descriptions in this chapter. Also see the *VISA Resource Manager* description in Appendix B, *Resources*.

viOpenDefaultRM

Purpose

This function returns a session to the Default Resource Manager resource.

C Syntax

```
ViStatus viOpenDefaultRM(ViPSession sesn)
```

Visual Basic Syntax

```
viOpenDefaultRM&(sesn&)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
sesn	OUT	Unique logical identifier to a Default Resource Manager session.

Return Values

Completion Codes	Description
VI_SUCCESS	Session to the Default Resource Manager resource created successfully.
VI_WARN_CONFIG_NLOADED	At least one configured Passport module could not be loaded.

Error Codes	Description
VI_ERROR_SYSTEM_ERROR	The VISA system failed to initialize.
VI_ERROR_ALLOC	Insufficient system resources to create a session to the Default Resource Manager resource.

Error Codes	Description
VI_ERROR_INV_SETUP	Some implementation-specific configuration file is corrupt or does not exist.
VI_ERROR_LIBRARY_NFOUND	A code library required by VISA could not be located or loaded.

Description

The viOpenDefaultRM() function must be called before any VISA operations can be invoked. The first call to this function initializes the VISA system, including the Default Resource Manager resource, and also returns a session to that resource. Subsequent calls to this function return unique sessions to the same Default Resource Manager resource.

When a Resource Manager session is passed to viClose(), not only is that session closed, but also all find lists and device sessions (which that Resource Manager session was used to create) are closed.

Related Items

See the viOpen(), viClose(), and viFindRsrc() descriptions in this chapter. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

viOut8/viOut16/viOut32

Purpose

Writes an 8-bit, 16-bit, or 32-bit value to the specified memory space and offset.

C Syntax

```
ViStatus viOut8(ViSession vi, ViUInt16 space, ViBusAddress offset,
ViUInt8 val8)

ViStatus viOut16(ViSession vi, ViUInt16 space, ViBusAddress offset,
ViUInt16 val16)

ViStatus viOut32(ViSession vi, ViUInt16 space, ViBusAddress offset,
ViUInt32 val32)
```

Visual Basic Syntax

```
viOut8&(ByVal vi&, ByVal space%, ByVal offset&, ByVal val8 as Byte)

viOut16&(ByVal vi&, ByVal space%, ByVal offset&, ByVal val16%)

viOut32&(ByVal vi&, ByVal space%, ByVal offset&, ByVal val32&)
```

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, PXI INSTR, VXI INSTR, VXI MEMACC

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
space	IN	Specifies the address space. Refer to the table included in the *Description* section for more information.
offset	IN	Offset (in bytes) of the address or register to which to read.
val8, **val16**, or **val32**	IN	Data to write to bus (8 bits for `viOut8()`, 16 bits for `viOut16()`, and 32 bits for `viOut32()`).

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SPACE	Invalid address space specified.
VI_ERROR_INV_OFFSET	Invalid offset specified.
VI_ERROR_NSUP_OFFSET	Specified offset is not accessible from this hardware.
VI_ERROR_NSUP_WIDTH	Specified width is not supported by this hardware.
VI_ERROR_NSUP_ALIGN_OFFSET	The specified offset is not properly aligned for the access width of the operation.
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).

Description

The `viOut`*`XX`*`()` operations use the specified address space to write 8, 16, or 32 bits of data, respectively, to the specified offset. These operations do not require `viMapAddress()` to be called prior to their invocation.

The following table lists the valid entries for specifying address space.

Value	Description

VXI, VME, and GPIB-VXI	`VI_A16_SPACE (1)`
	`VI_A24_SPACE (2)`
	`VI_A32_SPACE (3)`

PXI	`VI_PXI_CFG_SPACE (10)` `VI_PXI_BAR0_SPACE (11)` to `VI_PXI_BAR5_SPACE (16)`

INSTR Specific

Notice that **offset** specified in the `viOut8()`, `viOut16()`, and `viOut32()` operations for an INSTR Resource is the offset address relative to the device's allocated address base for the corresponding address space that was specified. For example, if **space** specifies `VI_A16_SPACE`, then **offset** specifies the offset from the logical address base address of the specified VXI device. If **space** specifies `VI_A24_SPACE` or `VI_A32_SPACE`, then **offset** specifies the offset from the base address of the VXI device's memory space allocated by the VXI Resource Manager within VXI A24 or A32 space.

MEMACC Specific

For a MEMACC Resource, the **offset** parameter specifies an absolute address.

Related Items

See the `viIn8/viIn16/viIn32()` descriptions in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

viParseRsrc

Purpose

Parse a resource string to get the interface information.

C Syntax

```
ViStatus viParseRsrc(ViSession sesn, ViRsrc rsrcName,
ViPUInt16 intfType, ViPUInt16 intfNum)
```

Visual Basic Syntax

```
viParseRsrc&(ByVal sesn&, ByVal rsrcName$, intfType%, intfNum%)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
sesn	IN	Resource Manager session (should always be the Default Resource Manager for VISA returned from `viOpenDefaultRM()`).
rsrcName	IN	Unique symbolic name of a resource.
intfType	OUT	Interface type of the given resource string.
intfNum	OUT	Board number of the interface of the given resource string.

Return Values

Completion Codes	Description
VI_SUCCESS	Resource string is valid.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **sesn** does not support this operation. For VISA, this operation is supported only by the Default Resource Manager session.
VI_ERROR_INV_RSRC_NAME	Invalid resource reference specified. Parsing error.
VI_ERROR_RSRC_NFOUND	Insufficient location information or resource not present in the system.
VI_ERROR_ALLOC	Insufficient system resources to parse the string.
VI_ERROR_LIBRARY_NFOUND	A code library required by VISA could not be located or loaded.
VI_ERROR_INTF_NUM_NCONFIG	The interface type is valid, but the specified interface number is not configured.

Description

This operation parses a resource string to verify its validity. It should succeed for all strings returned by `viFindRsrc()` and recognized by `viOpen()`. This operation is useful if you want to know what interface a given resource descriptor would use without actually opening a session to it.

The values returned in **intfType** and **intfNum** correspond to the attributes `VI_ATTR_INTF_TYPE` and `VI_ATTR_INTF_NUM`. These values would be the same if a user opened that resource with `viOpen()` and queried the attributes with `viGetAttribute()`.

Calling `viParseRsrc()` with "`VXI::1::INSTR`" will produce the same results as invoking it with "`vxi::1::instr`".

Related Items

See the `viFindRsrc()` and `viOpen()` descriptions in this chapter. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

viPeek8/viPeek16/viPeek32

Purpose

Reads an 8-bit, 16-bit, or 32-bit value from the specified address.

C Syntax

```
void viPeek8(ViSession vi, ViAddr addr, ViPUInt8 val8)
void viPeek16(ViSession vi, ViAddr addr, ViPUInt16 val16)
void viPeek32(ViSession vi, ViAddr addr, ViPUInt32 val32)
```

Visual Basic Syntax

```
viPeek8(ByVal vi&, ByVal addr&, val8 as Byte)
viPeek16(ByVal vi&, ByVal addr&, val16%)
viPeek32(ByVal vi&, ByVal addr&, val32&)
```

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, PXI INSTR, VXI INSTR, VXI MEMACC

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
addr	IN	Source address to read the value.
val8, **val16**, or **val32**	OUT	Data read from bus (8 bits for `viPeek8()`, 16 bits for `viPeek16()`, and 32 bits for `viPeek32()`).

Return Values

None

Description

The `viPeek`*XX*`()` operations read an 8-bit, 16-bit, or 32-bit value, respectively, from the address location specified in **addr**. The address must be a valid memory address in the current process mapped by a previous `viMapAddress()` call.

Related Items

See the `viMapAddress()` and `viPoke8/viPoke16/viPoke32()` descriptions. See the `VI_ATTR_WIN_ACCESS` description in Chapter 3, *Attributes*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

viPoke8/viPoke16/viPoke32

Purpose

Writes an 8-bit, 16-bit, or 32-bit value to the specified address.

C Syntax

```
void viPoke8(ViSession vi, ViAddr addr, ViUInt8 val8)
void viPoke16(ViSession vi, ViAddr addr, ViUInt16 val16)
void viPoke32(ViSession vi, ViAddr addr, ViUInt32 val32)
```

Visual Basic Syntax

```
viPoke8(ByVal vi&, ByVal addr&, ByVal val8 as Byte)
viPoke16(ByVal vi&, ByVal addr&, ByVal val16%)
viPoke32(ByVal vi&, ByVal addr&, ByVal val32&)
```

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, PXI INSTR, VXI INSTR, VXI MEMACC

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
addr	IN	Destination address to store the value.
val8, **val16**, or **val32**	IN	Value to be stored (8 bits for `viPoke8()`, 16 bits for `viPoke16()`, and 32 bits for `viPoke32()`).

Return Values

None

Description

The `viPoke`*XX*`()` operations store the content of an 8-bit, 16-bit, or 32-bit value, respectively, to the address pointed to by **addr**. The address must be a valid memory address in the current process mapped by a previous `viMapAddress()` call.

Related Items

See the `viMapAddress()` and `viPeek8/viPeek16/viPeek32()` descriptions. See the `VI_ATTR_WIN_ACCESS` description in Chapter 3, *Attributes*. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

viPrintf

Purpose

Converts, formats, and sends the parameters (designated by**...**) to the device as specified by the format string.

C Syntax

```
ViStatus viPrintf(ViSession vi, ViString writeFmt, ...)
```

Visual Basic Syntax

N/A

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
writeFmt	IN	String describing the format for arguments.
…	IN	Parameters to which the format string is applied.

Return Values

Completion Codes	Description
VI_SUCCESS	Parameters were successfully formatted.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_IO	Could not perform write operation because of I/O error.
VI_ERROR_TMO	Timeout expired before write operation completed.
VI_ERROR_INV_FMT	A format specifier in the **writeFmt** string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the **writeFmt** string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

The `viPrintf()` operation sends data to a device as specified by the format string. Before sending the data, the operation formats the arguments in the parameter list as specified in the **writeFmt** string. The `viWrite()` operation performs the actual low-level I/O to the device. As a result, you should not use the `viWrite()` and `viPrintf()` operations in the same session.

The **writeFmt** string can include regular character sequences, special formatting characters, and special format specifiers. The regular characters (including white spaces) are written to the device unchanged. The special characters consist of '\' (backslash) followed by a character. The format specifier sequence consists of '%' (percent) followed by an optional modifier (flag), followed by a format code.

Special Formatting Characters

Special formatting character sequences send special characters. The following table lists the special characters and describes what they send to the device.

Formatting Character	Character Sent to Device
\n	Sends the ASCII LF character. The END identifier will also be automatically sent.
\r	Sends an ASCII CR character.
\t	Sends an ASCII TAB character.
\###	Sends the ASCII character specified by the octal value.
\x##	Sends the ASCII character specified by the hexadecimal value.
\"	Sends the ASCII double-quote (") character.
\\	Sends a backslash (\) character.

Format Specifiers

The format specifiers convert the next parameter in the sequence according to the modifier and format code, after which the formatted data is written to the specified device. The format specifier takes the following syntax:

%[modifiers]*format code*

where *format code* specifies which data type the argument is represented in. Modifiers are optional codes that describe the target data.

In the following tables, a ‘d’ format code refers to all conversion codes of type *integer* (‘d’, ‘i’, ‘o’, ‘u’, ‘x’, ‘X’), unless specified as %d only. Similarly, an ‘f’ format code refers to all conversion codes of type *float* (‘f’, ‘e’, ‘E’, ‘g’, ‘G’), unless specified as %f only.

Every conversion command starts with the% character and ends with a conversion character (format code). Between the% character and the format code, the following modifiers can appear in the sequence.

ANSI C Standard Modifiers

Modifier	Supported with Format Code	Description
An integer specifying *field width*.	`d`, `f`, `s` format codes	This specifies the minimum field width of the converted argument. If an argument is shorter than the *field width*, it will be padded on the left (or on the right if the - flag is present). Special case: For the `@H`, `@Q`, and `@B` flags, the *field width* includes the `#H`, `#Q`, and `#B` strings, respectively. An asterisk (`*`) may be present in lieu of a field width modifier, in which case an extra **arg** is used. This **arg** must be an integer representing the *field width*.
An integer specifying *precision*.	`d`, `f`, `s` format codes	The *precision* string consists of a string of decimal digits. A decimal point (`.`) must prefix the *precision* string. The *precision* string specifies the following: a. The minimum number of digits to appear for the `@1`, `@H`, `@Q`, and `@B` flags and the `i`, `o`, `u`, `x`, and `X` format codes. b. The maximum number of digits after the decimal point in case of `f` format codes. c. The maximum numbers of characters for the string (`s`) specifier. d. Maximum significant digits for `g` format code. An asterisk (`*`) may be present in lieu of a *precision* modifier, in which case an extra **arg** is used. This **arg** must be an integer representing the *precision* of a numeric field.

Modifier	Supported with Format Code	Description
An argument length modifier. h, l, L, z, and Z are legal values. (z and Z are not ANSI C standard modifiers.)	h (d, b, B format codes) l (d, f, b, B format codes) L (f format code) z (b, B format codes) Z (b, B format codes)	The argument length modifiers specify one of the following: a. The h modifier promotes the argument to a short or unsigned short, depending on the format code type. b. The l modifier promotes the argument to a long or unsigned long. c. The L modifier promotes the argument to a long double parameter. d. The z modifier promotes the argument to an array of floats. e. The Z modifier promotes the argument to an array of doubles.

Enhanced Modifiers to ANSI C Standards

Modifier	Supported with Format Code	Description
A comma (,) followed by an integer *n*, where *n* represents the array size.	%d (plus variants) and %f only	The corresponding argument is interpreted as a reference to the first element of an array of size *n*. The first *n* elements of this list are printed in the format specified by the format code. An asterisk (*) may be present after the comma (,) modifier, in which case an extra **arg** is used. This **arg** must be an integer representing the array size of the given type.

Modifier	Supported with Format Code	Description
@1	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined NR1 compatible number, which is an integer without any decimal point (for example, 123).
@2	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined NR2 compatible number. The NR2 number has at least one digit after the decimal point (for example, 123.45).
@3	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined NR3 compatible number. An NR3 number is a floating point number represented in an exponential form (for example, 1.2345E-67).
@H	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined <HEXADECIMAL NUMERIC RESPONSE DATA>. The number is represented in a base of sixteen form. Only capital letters should represent numbers. The number is of form #H*XXX*.., where *XXX*.. is a hexadecimal number (for example, #HAF35B).
@Q	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined <OCTAL NUMERIC RESPONSE DATA>. The number is represented in a base of eight form. The number is of the form #Q*YYY*.., where *YYY*.. is an octal number (for example, #Q71234).
@B	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined <BINARY NUMERIC RESPONSE DATA>. The number is represented in a base two form. The number is of the form #B*ZZZ*.., where *ZZZ*.. is a binary number (for example, #B011101001).

The following are the allowed format code characters. A format specifier sequence should include one and only one format code.

ANSI C Standard Format Codes

% Send the ASCII percent (%) character.

c Argument type: A character to be sent.

`d` Argument type: An integer.

Modifier	Interpretation
Default functionality	Print an integer in NR1 format (an integer without a decimal point).
`@2` or `@3`	The integer is converted into a floating point number and output in the correct format.
field width	Minimum field width of the output number. Any of the six IEEE 488.2 modifiers can also be specified with *field width*.
Length modifier `l`	**arg** is a long integer.
Length modifier `h`	**arg** is a short integer.
`,` *array size*	**arg** points to an array of integers (or long or short integers, depending on the length modifier) of size array size. The elements of this array are separated by array size - 1 commas and output in the specified format.

`f` Argument type: A floating point number.

Modifier	Interpretation
Default functionality	Print a floating point number in NR2 format (a number with at least one digit after the decimal point).
`@1`	Print an integer in NR1 format. The number is truncated.
`@3`	Print a floating point number in NR3 format (scientific notation). *Precision* can also be specified.
field width	Minimum field width of the output number. Any of the six IEEE 488.2 modifiers can also be specified with *field width*.
Length modifier `l`	**arg** is a double float.
Length modifier `L`	**arg** is a long double.
`,` *array size*	**arg** points to an array of floats (or doubles or long doubles, depending on the length modifier) of size array size. The elements of this array are separated by array size - 1 commas and output in the specified format.

`s` Argument type: A reference to a NULL-terminated string that is sent to the device without change.

Enhanced Format Codes

b Argument type: A location of a block of data.

Flag or Modifier	Interpretation
Default functionality	The data block is sent as an IEEE 488.2 <DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA>. A count (long integer) must appear as a flag that specifies the number of elements (by default, bytes) in the block. A *field width* or *precision* modifier is not allowed with this format code.
* (asterisk)	An asterisk may be present instead of the count. In such a case, two **args** are used, the first of which is a long integer specifying the count of the number of elements in the data block. The second **arg** is a reference to the data block. The size of an element is determined by the optional length modifier (see below), and the default is byte width.
Length modifier h	**arg** points to an array of unsigned short integers (16 bits). The count corresponds to the number of words rather than bytes. The data is swapped and padded into standard IEEE 488.2 format, if native computer representation is different.
Length modifier l	**arg** points to an array of unsigned long integers. The count specifies the number of longwords (32 bits). Each longword data is swapped and padded into standard IEEE 488.2 format, if native computer representation is different.
Length modifier z	**arg** points to an array of floats. The count specifies the number of floating point numbers (32 bits). The numbers are represented in IEEE 754 format, if native computer representation is different.
Length modifier Z	**arg** points to an array of doubles. The count specifies the number of double floats (64 bits). The numbers will be represented in IEEE 754 format, if native computer representation is different.

B Argument type: A location of a block of data. The functionality is similar to b**,** except the data block is sent as an IEEE 488.2 <INDEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA>. This format involves sending an ASCII LF character with the END indicator set after the last byte of the block.

The END indicator is not appended when LF(\n) is part of a binary data block, as with %b or %B.

y Argument type: A location of a block of binary data.

Modifier	Interpretation
Default functionality	The data block is sent as raw binary data. A count (long integer) must appear as a flag that specifies the number of elements (by default, bytes) in the block. A field width or precision modifier is not allowed with this format code.
* (asterisk)	An asterisk may be present instead of the count. In such a case, two **arg**s are used, the first of which is a long integer specifying the count of the number of elements in the data block. The second **arg** is a reference to the data block. The size of an element is determined by the optional length modifier (see below), and the default is byte width.
Length modifier `h`	**arg** points to an array of unsigned short integers (16 bits). The count corresponds to the number of words rather than bytes. If the optional `!ol` byte order modifier is present, the data is sent in little endian format; otherwise, the data is sent in standard IEEE 488.2 format. The data will be byte swapped and padded as appropriate if native computer representation is different.
Length modifier `l`	**arg** points to an array of unsigned long integers (32 bits). The count specifies the number of longwords rather than bytes. If the optional `!ol` byte order modifier is present, the data is sent in little endian format; otherwise, the data is sent in standard IEEE 488.2 format. The data will be byte swapped and padded as appropriate if native computer representation is different.
Byte order modifier `!ob`	Data is sent in standard IEEE 488.2 (big endian) format. This is the default behavior if neither `!ob` nor `!ol` is present.
Byte order modifier `!ol`	Data is sent in little endian format.

Other ANSI C Conversion Codes

For ANSI C compatibility, VISA also supports the following conversion codes for output codes: ‘i’, ‘o’, ‘u’, ‘n’, ‘x’, ‘X’, ‘e’, ‘E’, ‘g’, ‘G’, and ‘p.’ For further explanation of these conversion codes, see the ANSI C Standard.

Also refer to your ANSI C documentation for information on the `printf` function.

Note VISA will not send out the data across the bus, by default, until a '\n' character is encountered in the format string (not the data stream). You can modify this behavior with the `VI_ATTR_WR_BUF_OPER_MODE` attribute or with the `viFlush()` operation.

Related Items

See the `viFlush()`, `viSPrintf()`, `viVPrintf()`, and `viVSPrintf()`, and `viScanf()`, descriptions in this chapter, and `VI_ATTR_WR_BUF_OPER_MODE` in Chapter 3, *Attributes*. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*. Also refer to your ANSI C documentation for information on the `printf` function.

viQueryf

Purpose

Performs a formatted write and read through a single call to an operation.

C Syntax

```
ViStatus viQueryf(ViSession vi, ViString writeFmt,
ViString readFmt,...)
```

Visual Basic Syntax

N/A

Resource Classes

GPIB INSTR, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
writeFmt	IN	String describing the format of write arguments.
readFmt	IN	String describing the format of read arguments.
…	IN/OUT	Parameters to which write and read format strings are applied.

Return Values

Completion Codes	Description
VI_SUCCESS	Successfully completed the query operation.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_IO	Could not perform read/write operation because of I/O error.

Error Codes	Description
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_TMO	Timeout occurred before read/write operation completed.
VI_ERROR_INV_FMT	A format specifier in the **writeFmt** or **readFmt** string is invalid.
VI_ERROR_NSUP_FMT	The format specifier is not supported for current argument type.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

This operation provides a mechanism of *Send, then receive* typical to a command sequence from a commander device. In this manner, the response generated from the command can be read immediately.

This operation is a combination of the `viPrintf()` and `viScanf()` operations. The first *n* arguments corresponding to the first format string are formatted by using the **writeFmt** string, then sent to the device. The write buffer is flushed immediately after the write portion of the operation completes. After these actions, the response data is read from the device into the remaining parameters (starting from parameter *n*+1) using the **readFmt** string.

Note Because the prototype for this function cannot provide complete type-checking, remember that all output parameters must be passed by reference.

Related Items

See the `viPrintf()`, `viScanf()`, and `viVQueryf()` descriptions in this chapter. Also see the *INSTR Resource* and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viRead

Purpose

Reads data from device or interface synchronously.

C Syntax

```
ViStatus viRead(ViSession vi, ViPBuf buf, ViUInt32 count,
ViPUInt32 retCount)
```

Visual Basic Syntax

```
viRead&(ByVal vi&, ByVal buf$, ByVal count&, retCount&)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	OUT	Location of a buffer to receive data from device.
count	IN	Number of bytes to be read.
retCount	OUT	Number of bytes actually transferred.

Return Values

Completion Codes	Description
VI_SUCCESS	The operation completed successfully and the END indicator was received (for interfaces that have END indicators). This completion code is returned regardless of whether the termination character is received or the number of bytes read is equal to **count**.
VI_SUCCESS_TERM_CHAR	The specified termination character was read but no END indicator was received. This completion code is returned regardless of whether the number of bytes read is equal to **count**.
VI_SUCCESS_MAX_CNT	The number of bytes read is equal to **count**. No END indicator was received and no termination character was read.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_OUTP_PROT_VIOL	Device reported an output protocol error during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SETUP	Unable to start read operation because setup is invalid (due to attributes being set to an inconsistent state).

Error Codes	Description
VI_ERROR_NCIC	The interface associated with the given **vi** is not currently the controller in charge.
VI_ERROR_NLISTENERS	No-listeners condition is detected (both NRFD and NDAC are unasserted).
VI_ERROR_ASRL_PARITY	A parity error occurred during transfer.
VI_ERROR_ASRL_FRAMING	A framing error occurred during transfer.
VI_ERROR_ASRL_OVERRUN	An overrun error occurred during transfer. A character was not read from the hardware before the next character arrived.
VI_ERROR_IO	An unknown I/O error occurred during transfer.
VI_ERROR_CONN_LOST	The I/O connection for the given session has been lost.

Description

The viRead() operation synchronously transfers data. The data read is to be stored in the buffer represented by **buf**. This operation returns only when the transfer terminates. Only one synchronous read operation can occur at any one time.

Related Items

See the viReadAsync(), viBufRead(), viReadToFile(), and viWrite() descriptions in this chapter. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viReadAsync

Purpose

Reads data from device or interface asynchronously.

C Syntax

```
ViStatus viReadAsync(ViSession vi, ViPBuf buf, ViUInt32 count,
ViPJobId jobId)
```

Visual Basic Syntax

N/A

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	OUT	Location of a buffer to receive data from device.
count	IN	Number of bytes to be read.
jobId	OUT	Job ID of this asynchronous read operation.

Return Values

Completion Codes	Description
VI_SUCCESS	Asynchronous read operation successfully queued.
VI_SUCCESS_SYNC	Read operation performed synchronously.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_QUEUE_ERROR	Unable to queue read operation.
VI_ERROR_IN_PROGRESS	Unable to queue the asynchronous operation because there is already an operation in progress.

Description

The viReadAsync() operation asynchronously transfers data. The data read is to be stored in the buffer represented by **buf**. This operation normally returns before the transfer terminates.

Before calling this operation, you should enable the session for receiving I/O completion events. After the transfer has completed, an I/O completion event is posted.

The operation returns **jobId**, which you can use with either viTerminate() to abort the operation, or with an I/O completion event to identify which asynchronous read operation completed. VISA will never return VI_NULL for a valid **jobID**.

Related Items

See the viEnableEvent(), viRead(), viTerminate(), and viWriteAsync() descriptions in this chapter. See the VI_EVENT_IO_COMPLETION description in Chapter 4, *Events*. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viReadSTB

Purpose

Reads a status byte of the service request.

C Syntax

```
ViStatus viReadSTB(ViSession vi, ViPUInt16 status)
```

Visual Basic Syntax

```
viReadSTB&(ByVal vi&, status%)
```

Resource Classes

GPIB INSTR, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
status	OUT	Service request status byte.

Return Values

Completion Codes	Description
VI_SUCCESS	The operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_SRQ_NOCCURRED	Service request has not been received for the session.

Error Codes	Description
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_NCIC	The interface associated with the given **vi** is not currently the controller in charge.
VI_ERROR_NLISTENERS	No-listeners condition is detected (both NRFD and NDAC are unasserted).
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_CONN_LOST	The I/O connection for the given session has been lost.

Description

The viReadSTB() operation reads a service request status from a service requester (the message-based device). For example, on the IEEE 488.2 interface, the message is read by polling devices; for other types of interfaces, a message is sent in response to a service request to retrieve status information. For a session to a Serial device or Ethernet socket, if VI_ATTR_IO_PROT is VI_PROT_4882_STRS, the device is sent the string "*STB?\n", and then the device's status byte is read; otherwise, this operation is not valid. If the status information is only one byte long, the most significant byte is returned with the zero value. If the service requester does not respond in the actual timeout period, VI_ERROR_TMO is returned.

Related Items

See the VI_ATTR_IO_PROT description in Chapter 3, *Attributes*. See the VI_EVENT_SERVICE_REQ description in Chapter 4, *Events*. Also see the *INSTR Resource* and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viReadToFile

Purpose

Read data synchronously, and store the transferred data in a file.

C Syntax

```
ViStatus viReadToFile(ViSession vi, ViString fileName,
ViUInt32 count, ViPUInt32 retCount)
```

Visual Basic Syntax

```
viReadToFile& (ByVal vi&, ByVal filename$, ByVal count&, retCount&)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
fileName	IN	Name of file to which data will be written.
count	IN	Number of bytes to be read.
retCount	OUT	Number of bytes actually transferred.

Return Values

Completion Codes	Description
VI_SUCCESS	The operation completed successfully and the END indicator was received (for interfaces that have END indicators).
VI_SUCCESS_TERM_CHAR	The specified termination character was read.
VI_SUCCESS_MAX_CNT	The number of bytes read is equal to count.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_OUTP_PROT_VIOL	Device reported an output protocol error during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SETUP	Unable to start read operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_NCIC	The interface associated with the given **vi** is not currently the controller in charge.
VI_ERROR_NLISTENERS	No listeners condition is detected (both NRFD and NDAC are deasserted).
VI_ERROR_ASRL_PARITY	A parity error occurred during transfer.
VI_ERROR_ASRL_FRAMING	A framing error occurred during transfer.
VI_ERROR_ASRL_OVERRUN	An overrun error occurred during transfer. A character was not read from the hardware before the next character arrived.
VI_ERROR_IO	An unknown I/O error occurred during transfer.
VI_ERROR_FILE_ACCESS	An error occurred while trying to open the specified file. Possible reasons include an invalid path or lack of access rights.

Error Codes	Description
VI_ERROR_FILE_IO	An error occurred while accessing the specified file.
VI_ERROR_CONN_LOST	The I/O connection for the given session has been lost.

Description

This read operation synchronously transfers data. The file specified in fileName is opened in binary write-only mode. If the value of VI_ATTR_FILE_APPEND_EN is VI_FALSE, any existing contents are destroyed; otherwise, the file contents are preserved. The data read is written to the file. This operation returns only when the transfer terminates.

This operation is useful for storing raw data to be processed later.

Special Values for retCount Parameter

Value	Action Description
VI_NULL	Do not return the number of bytes transferred.

Related Items

See the viRead(), and viWriteFromFile() descriptions in this chapter and VI_ATTR_FILE_APPEND_EN in Chapter 3, *Attributes*. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viScanf

Purpose

Reads, converts, and formats data using the format specifier. Stores the formatted data in the parameters (designated by **...**).

C Syntax

```
ViStatus viScanf(ViSession vi, ViString readFmt, ...)
```

Visual Basic Syntax

N/A

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
readFmt	IN	String describing the format for arguments.
…	OUT	Parameters into which the data is read and the format string is applied.

Return Values

Completion Codes	Description
VI_SUCCESS	Data was successfully read and formatted into **...** parameter(s).

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.

Error Codes	Description
VI_ERROR_IO	Could not perform read operation because of I/O error.
VI_ERROR_TMO	Timeout expired before read operation completed.
VI_ERROR_INV_FMT	A format specifier in the **readFmt** string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the **readFmt** string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

The viScanf() operation receives data from a device, formats it by using the format string, and stores the resulting data in the **arg** parameter list. The viRead() operation is used for the actual low-level read from the device. As a result, you should not use the viRead() and viScanf() operations in the same session.

Note Because the prototype for this function cannot provide complete type-checking, remember that all output parameters must be passed by reference.

The format string can have format specifier sequences, white characters, and ordinary characters. The white characters—blank, vertical tabs, horizontal tabs, form feeds, new line/linefeed, and carriage return—are ignored except in the case of %c and %[ ]. All other ordinary characters except % should match the next character read from the device.

The format string consists of a %, followed by optional modifier flags, followed by one of the format codes in that sequence. It is of the form:

%[modifier]format code

where the optional modifier describes the data format, while format code indicates the nature of data (data type). One and only one format code should be performed at the specifier sequence. A format specification directs the conversion to the next input **arg**.

The results of the conversion are placed in the variable that the corresponding argument points to, unless the * assignment-suppressing character is given. In such a case, no **arg** is used and the results are ignored.

The viScanf() operation accepts input until an END indicator is read or all the format specifiers in the **readFmt** string are satisfied. Thus, detecting an END indicator before the **readFmt** string is fully consumed will result in ignoring the rest of the format string. Also, if

some data remains in the buffer after all format specifiers in the **readFmt** string are satisfied, the data will be kept in the buffer and will be used by the next `viScanf()` operation.

When `viScanf()` times out, the next call to `viScanf()` will read from an empty buffer and force a read from the device.

Notice that when an END indicator is received, not all arguments in the format string may be consumed. However, the operation still returns a successful completion code.

The following two tables describe optional modifiers that can be used in a format specifier sequence.

ANSI C Standard Modifiers

Modifier	Supported with Format Code	Description
An integer representing the *field width*	`%s`, `%c`, `%[]` format codes	It specifies the maximum field width that the argument will take. A '#' may also appear instead of the integer *field width*, in which case the next **arg** is a reference to the *field width*. This **arg** is a reference to an integer for `%c` and `%s`. The *field width* is not allowed for `%d` or `%f`.
A length modifier ('`h`,' '`l`,' '`L`,' '`z`,' or '`Z`'). `z` and `Z` are not ANSI C standard modifiers.	`h` (`d`, `b` format codes) `l` (`d`, `f`, `b` format codes) `L` (`f` format code) `z` (`b` format code) `Z` (`b` format code)	The argument length modifiers specify one of the following: a. The `h` modifier promotes the argument to be a reference to a short integer or unsigned short integer, depending on the format code. b. The `l` modifier promotes the argument to point to a long integer or unsigned long integer. c. The `L` modifier promotes the argument to point to a long double floats parameter. d. The `z` modifier promotes the argument to point to an array of floats. e. The `Z` modifier promotes the argument to point to an array of double floats.
*	All format codes	An asterisk (`*`) acts as the assignment suppression character. The input is not assigned to any parameters and is discarded.

Enhanced Modifiers to ANSI C Standards

Modifier	Supported with Format Code	Description
A comma (`,`) followed by an integer *n*, where *n* represents the array size.	`%d` (plus variants) and `%f` only	The corresponding argument is interpreted as a reference to the first element of an array of size *n*. The first *n* elements of this list are printed in the format specified by the format code. A number sign (#) may be present after the comma (`,`) modifier, in which case an extra **arg** is used. This **arg** must be an integer representing the array size of the given type.
`@1`	`%d` (plus variants) and `%f` only	Converts to an IEEE 488.2 defined NR1 compatible number, which is an integer without any decimal point (for example, 123).
`@2`	`%d` (plus variants) and `%f` only	Converts to an IEEE 488.2 defined NR2 compatible number. The NR2 number has at least one digit after the decimal point (for example, 123.45).
`@H`	`%d` (plus variants) and `%f` only	Converts to an IEEE 488.2 defined <HEXADECIMAL NUMERIC RESPONSE DATA>. The number is represented in a base of sixteen form. Only capital letters should represent numbers. The number is of form `#H`*`XXX`*.., where *`XXX`*.. is a hexadecimal number (for example, `#HAF35B`).
`@Q`	`%d` (plus variants) and `%f` only	Converts to an IEEE 488.2 defined <OCTAL NUMERIC RESPONSE DATA>. The number is represented in a base of eight form. The number is of the form `#Q`*`YYY`*.., where *`YYY`*.. is an octal number (for example, `#Q71234`).
`@B`	`%d` (plus variants) and `%f` only	Converts to an IEEE 488.2 defined <BINARY NUMERIC RESPONSE DATA>. The number is represented in a base two form. The number is of the form `#B`*`ZZZ`*.., where *`ZZZ`*.. is a binary number (for example, `#B011101001`).

ANSI C Standard Format Codes

c Argument type: A reference to a character.

Flags or Modifiers	Interpretation
Default functionality	A character is read from the device and stored in the parameter.
field width	*field width* number of characters are read and stored at the reference location (the default *field width* is 1). No NULL character is added at the end of the data block.

Note This format code does not ignore white space in the device input stream.

d Argument type: A reference to an integer.

Flags or Modifiers	Interpretation
Default functionality	Characters are read from the device until an entire number is read. The number read may be in either IEEE 488.2 formats <DECIMAL NUMERIC PROGRAM DATA>, also known as NRf; flexible numeric representation (NR1, NR2, NR3...); or <NON-DECIMAL NUMERIC PROGRAM DATA> (#H, #Q, and #B).
field width	The input number will be stored in a field at least this wide.
Length modifier l	**arg** is a reference to a long integer.
Length modifier h	**arg** is a reference to a short integer. Rounding is performed according to IEEE 488.2 rules (0.5 and up).
, *array size*	**arg** points to an array of integers (or long or short integers, depending on the length modifier) of size array size. The elements of this array should be separated by commas. Elements will be read until either array size number of elements are consumed or they are no longer separated by commas.

`f` Argument type: A reference to a floating point number.

Flags or Modifiers	Interpretation
Default functionality	Characters are read from the device until an entire number is read. The number read may be in either IEEE 488.2 formats <DECIMAL NUMERIC PROGRAM DATA> (NRf) or <NON-DECIMAL NUMERIC PROGRAM DATA> (`#H`, `#Q`, and `#B`)
field width	The input will be stored in a field at least this wide.
Length modifier `l`	**arg** is a reference to a double floating point number.
Length modifier `L`	**arg** is a reference to a long double number.
`,` *array size*	**arg** points to an array of floats (or double or long double, depending on the length modifier) of size array size. The elements of this array should be separated by commas. Elements will be read until either array size number of elements are consumed or they are no longer separated by commas.

`s` Argument type: A reference to a string.

Flags or Modifiers	Interpretation
Default functionality	All leading white space characters are ignored. Characters are read from the device into the string until a white space character is read.
field width	This flag gives the maximum string size. If the *field width* contains a number sign (#), two arguments are used. The first argument read is a pointer to an integer specifying the maximum array size. The second should be a reference to an array. In case of *field width* characters already read before encountering a white space, additional characters are read and discarded until a white space character is found. In case of # *field width*, the actual number of characters read are stored back in the integer pointed to by the first argument.

Enhanced Format Codes

b Argument type: A reference to a data array.

Flags or Modifiers	Interpretation
Default functionality	The data must be in IEEE 488.2 <ARBITRARY BLOCK PROGRAM DATA> format. The format specifier sequence should have a flag describing the *field width*, which will give a maximum count of the number of bytes (or words or longwords, depending on length modifiers) to be read from the device. If the *field width* contains a # sign, two arguments are used. The first **arg** read is a pointer to a long integer specifying the maximum number of elements that the array can hold. The second **arg** should be a reference to an array. Also, the actual number of elements read is stored back in the first argument. In absence of length modifiers, the data is assumed to be of byte-size elements. In some cases, data might be read until an END indicator is read.
Length modifier h	**arg** points to an array of 16-bit words, and count specifies the number of words. Data that is read is assumed to be in IEEE 488.2 byte ordering. It will be byte swapped and padded as appropriate to native computer format.
Length modifier l	**arg** points to an array of 32-bit longwords, and count specifies the number of longwords. Data that is read is assumed to be in IEEE 488.2 byte ordering. It will be byte swapped and padded as appropriate to native computer format.
Length modifier z	**arg** points to an array of floats, and count specifies the number of floating point numbers. Data that is read is an array of 32-bit IEEE 754 format floating point numbers.
Length modifier Z	**arg** points to an array of doubles, and the count specifies the number of floating point numbers. Data that is read is an array of 64-bit IEEE 754 format floating point numbers.

t Argument type: A reference to a string.

Flags or Modifiers	Interpretation
Default functionality	Characters are read from the device until the first END indicator is received. The character on which the END indicator was received is included in the buffer.
field width	This flag gives the maximum string size. If an END indicator is not received before *field width* number of characters, additional characters are read and discarded until an END indicator arrives. *#field width* has the same meaning as in %s.

T Argument type: A reference to a string.

Flags or Modifiers	Interpretation
Default functionality	Characters are read from the device until the first linefeed character (\n) is received. The linefeed character is included in the buffer.
field width	This flag gives the maximum string size. If a linefeed character is not received before *field width* number of characters, additional characters are read and discarded until a linefeed character arrives. *#field width* has the same meaning as in %s.

y Argument type: A location of a block of binary data.

Modifier	Interpretation
Default functionality	The data block is read as raw binary data. The format specifier sequence should have a flag describing the array size, which will give a maximum count of the number of bytes (or words or longwords, depending on length modifiers) to be read from the device. If the array size contains a # sign, two arguments are used. The first argument read is a pointer to a long integer that specifies the maximum number of elements that the array can hold. The second argument should be a reference to an array. Also, the actual number of elements read is stored back in the first argument. In absence of length modifiers, the data is assumed to be byte-size elements. In some cases, data might be read until an END indicator is read.
Length modifier `h`	The data block is assumed to be a reference to an array of unsigned short integers (16 bits). The count corresponds to the number of words rather than bytes. If the optional `!ol` modifier is present, the data read is assumed to be in little endian format; otherwise, the data read is assumed to be in standard IEEE 488.2 format. The data will be byte swapped and padded as appropriate to native computer format.
Length modifier `l`	The data block is assumed to be a reference to an array of unsigned long integers (32 bits). The count corresponds to the number of longwords rather than bytes. If the optional `!ol` modifier is present, the data read is assumed to be in little endian format; otherwise, the data read is assumed to be in standard IEEE 488.2 format. The data will be byte swapped and padded as appropriate to native computer format.
Byte order modifier `!ob`	The data being read is assumed to be in standard IEEE 488.2 (big endian) format. This is the default behavior if neither `!ob` nor `!ol` is present.
Byte order modifier `!ol`	The data being read is assumed to be in little endian format.

Other ANSI C Format Specifiers

For ANSI C compatibility, VISA also supports the following format specifiers for input codes: ‘`i`’, ‘`o`’, ‘`u`’, ‘`n`’, ‘`x`’, ‘`X`’, ‘`e`’, ‘`E`’, ‘`g`’, ‘`G`’, ‘`p`’, ‘`[...]`’, and ‘`[^...]`’. For further explanation of these conversion codes, see the ANSI C Standard.

Related Items

See the viFlush(), viPrintf(), viSScanf(), viVScanf(), and viVSScanf() descriptions in this chapter, and VI_ATTR_RD_BUF_OPER_MODE in Chapter 3, *Attributes*. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*. Also refer to your ANSI C documentation for information on the scanf function.

viSetAttribute

Purpose

Sets the state of an attribute.

C Syntax

```
ViStatus viSetAttribute(ViObject vi, ViAttr attribute,
ViAttrState attrState)
```

Visual Basic Syntax

```
viSetAttribute&(ByVal vi&, ByVal attribute&, ByVal attrState&)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
attribute	IN	Attribute for which the state is to be modified.
attrState	IN	The state of the attribute to be set for the specified object. The interpretation of the individual attribute value is defined by the object.

Return Values

Completion Codes	Description
VI_SUCCESS	Attribute value set successfully.
VI_WARN_NSUP_ATTR_STATE	Although the specified attribute state is valid, it is not supported by this implementation.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given object reference is invalid.
VI_ERROR_NSUP_ATTR	The specified attribute is not defined by the referenced object.
VI_ERROR_NSUP_ATTR_STATE	The specified state of the attribute is not valid, or is not supported as defined by the object.
VI_ERROR_ATTR_READONLY	The specified attribute is read-only.

Description

The viSetAttribute() operation is used to modify the state of an attribute for the specified object.

Both VI_WARN_NSUP_ATTR_STATE and VI_ERROR_NSUP_ATTR_STATE indicate that the specified attribute state is not supported. A resource normally returns the error code VI_ERROR_NSUP_ATTR_STATE when it cannot set a specified attribute state. The completion code VI_WARN_NSUP_ATTR_STATE is intended to alert the application that although the specified optional attribute state is not supported, the application should not fail. One example is attempting to set an attribute value that would increase performance speeds. This is different than attempting to set an attribute value that specifies required but nonexistent hardware (such as specifying a VXI ECL trigger line when no hardware support exists) or a value that would change assumptions a resource might make about the way data is stored or formatted (such as byte order).

Related Items

See the viGetAttribute() description in this chapter. Also see the *VISA Resource Template* description in Appendix B, *Resources*, and the attribute descriptions in Chapter 3, *Attributes*.

viSetBuf

Purpose

Sets the size for the formatted I/O and/or low-level I/O communication buffer(s).

C Syntax

```
ViStatus viSetBuf(ViSession vi, ViUInt16 mask, ViUInt32 size)
```

Visual Basic Syntax

```
viSetBuf&(ByVal vi&, ByVal mask%, ByVal size&)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
mask	IN	Specifies the type of buffer.
size	IN	The size to be set for the specified buffer(s).

Return Values

Completion Codes	Description
VI_SUCCESS	Buffer size set successfully.
VI_WARN_NSUP_BUF	The specified buffer is not supported.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.

Error Codes	Description
VI_ERROR_ALLOC	The system could not allocate the buffer(s) of the specified **size** because of insufficient resources.
VI_ERROR_INV_MASK	The system cannot set the buffer for the given **mask**.

Description

The viSetBuf() operation changes the buffer size of the read and/or write buffer for formatted I/O and/or serial communication. The **mask** parameter specifies the buffer for which to set the size. The **mask** parameter can specify multiple buffers by bit-ORing any of the following values together.

Flags	Interpretation
VI_READ_BUF (1)	Formatted I/O read buffer.
VI_WRITE_BUF (2)	Formatted I/O write buffer.
VI_IO_IN_BUF (16)	I/O communication receive buffer.
VI_IO_OUT_BUF (32)	I/O communication transmit buffer.

A call to viSetBuf() flushes the session's related read/write buffer(s). Although you can explicitly flush the buffers by making a call to viFlush(), the buffers are flushed implicitly under some conditions. These conditions vary for the viPrintf() and viScanf() operations.

Since not all serial drivers support user-defined buffer sizes, it is possible that a specific implementation of VISA may not be able to control this feature. If an application requires a specific buffer size for performance reasons, but a specific implementation of VISA cannot guarantee that size, then it is recommended to use some form of handshaking to prevent overflow conditions.

In previous versions of VISA, VI_IO_IN_BUF was known as VI_ASRL_IN_BUF and VI_IO_OUT_BUF was known as VI_ASRL_OUT_BUF.

Related Items

See the viFlush(), viPrintf(), and viScanf() descriptions in this chapter. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viSPrintf

Purpose

Converts, formats, and sends the parameters (designated by**...**) to a user-specified buffer as specified by the format string.

C Syntax

```
ViStatus viSPrintf(ViSession vi, ViPBuf buf, ViString writeFmt, ...)
```

Visual Basic Syntax

N/A

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	OUT	Buffer where data is to be written.
writeFmt	IN	The format string to apply to parameters in `ViVAList`.
...	IN	Parameters to which the format string is applied. The formatted data is written to the specified **buf**.

Return Values

Completion Codes	Description
`VI_SUCCESS`	Parameters were successfully formatted.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_INV_FMT	A format specifier in the **writeFmt** string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the **writeFmt** string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

The viSPrintf() operation is similar to viPrintf(), except that the output is not written to the device; it is written to the user-specified buffer. This output buffer will be NULL terminated.

If this operation outputs an END indicator before all the arguments are satisfied, then the rest of the **writeFmt** string is ignored and the buffer string is still terminated by a NULL.

Note The size of the **buf** parameter should be large enough to hold the formatted I/O contents plus the NULL termination character.

Related Items

See the viPrintf(), viSScanf(), viVPrintf(), and viVSPrintf() descriptions in this chapter. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viSScanf

Purpose

Reads, converts, and formats data from a user-specified buffer using the format specifier. Stores the formatted data in the parameters (designated by **...**).

C Syntax

```
ViStatus viSScanf(ViSession vi, ViBuf buf, ViString readFmt, ...)
```

Visual Basic Syntax

N/A

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	IN	Buffer from which data is read and formatted.
readFmt	IN	String describing the format for arguments.
...	OUT	Parameters into which the data is read and the format string is applied.

Return Values

Completion Codes	Description
VI_SUCCESS	Data was successfully read and formatted into **...** parameter(s).

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_INV_FMT	A format specifier in the **readFmt** string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the **readFmt** string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

The viSScanf() operation is similar to viScanf(), except that the data is read from a user-specified buffer rather than from a device.

Related Items

See the viScanf(), viSPrintf(), viVScanf(), and viVSScanf() descriptions in this chapter. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viStatusDesc

Purpose

Returns a user-readable description of the status code passed to the operation.

C Syntax

```
ViStatus viStatusDesc(ViObject vi, ViStatus status, ViChar desc[])
```

Visual Basic Syntax

```
viStatusDesc&(ByVal vi&, ByVal status&, ByVal desc$)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
status	IN	Status code to interpret.
desc	OUT	The user-readable string interpretation of the status code passed to the operation.

Return Values

Completion Codes	Description
VI_SUCCESS	Description successfully returned.
VI_WARN_UNKNOWN_STATUS	The status code passed to the operation could not be interpreted.

Description

The `viStatusDesc()` operation is used to retrieve a user-readable string that describes the status code presented. If the string cannot be interpreted, the operation returns the warning code `VI_WARN_UNKNOWN_STATUS`. However, the output string **desc** is valid regardless of the status return value.

Note The size of the **desc** parameter should be at least 256 bytes.

Related Items

See Appendix A, *Status Codes*, for a complete list of the possible status codes for each operation. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

viTerminate

Purpose

Requests a VISA session to terminate normal execution of an operation.

C Syntax

```
ViStatus viTerminate(ViObject vi, ViUInt16 degree, ViJobId jobId))
```

Visual Basic Syntax

N/A

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
degree	IN	VI_NULL (0).
jobId	IN	Specifies an operation identifier.

Return Values

Completion Codes	Description
VI_SUCCESS	Request serviced successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given object reference is invalid.
VI_ERROR_INV_JOB_ID	Specified job identifier is invalid.
VI_ERROR_INV_DEGREE	Specified **degree** is invalid.

Description

This operation is used to request a session to terminate normal execution of an operation, as specified by the **jobId** parameter. The **jobId** parameter is a unique value generated from each call to an asynchronous operation.

If a user passes `VI_NULL` as the **jobId** value to `viTerminate()`, VISA will abort any calls in the current process executing on the specified **vi**. Any call that is terminated this way should return `VI_ERROR_ABORT`. Due to the nature of multi-threaded systems, for example where operations in other threads may complete normally before the operation `viTerminate()` has any effect, the specified return value is not guaranteed.

Related Items

See the `viReadAsync()`, `viWriteAsync()` and `viMoveAsync()` descriptions in this chapter. See the `VI_EVENT_IO_COMPLETION` description in Chapter 4, *Events*. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

viUninstallHandler

Purpose

Uninstalls handlers for events.

C Syntax

```
ViStatus viUninstallHandler(ViSession vi, ViEventType eventType,
ViHndlr handler, ViAddr userHandle)
```

Visual Basic Syntax

N/A

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
eventType	IN	Logical event identifier.
handler	IN	Interpreted as a valid reference to a handler to be uninstalled by a client application.
userHandle	IN	A value specified by an application that can be used for identifying handlers uniquely in a session for an event.

Return Values

Completion Codes	Description
VI_SUCCESS	Event handler successfully uninstalled.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_INV_EVENT	Specified event type is not supported by the resource.
VI_ERROR_INV_HNDLR_REF	Either the specified handler reference or the user context value (or both) does not match any installed handler.
VI_ERROR_HNDLR_NINSTALLED	A handler is not currently installed for the specified event.

Description

The viUninstallHandler() operation allows applications to uninstall handlers for events on sessions. Applications should also specify the value in the **userHandle** parameter that was passed while installing the handler. VISA identifies handlers uniquely using the handler reference and this value. All the handlers, for which the handler reference and the value matches, are uninstalled. Specifying VI_ANY_HNDLR as the value for the **handler** parameter causes the operation to uninstall all the handlers with the matching value in the **userHandle** parameter.

Related Items

See the viInstallHandler() description in this chapter. Also see the viEventHandler() description for its parameter description. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

viUnlock

Purpose

Relinquishes a lock for the specified resource.

C Syntax

```
ViStatus viUnlock(ViSession vi)
```

Visual Basic Syntax

```
viUnlock&(ByVal vi&)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.

Return Values

Completion Codes	Description
VI_SUCCESS	Lock successfully relinquished.
VI_SUCCESS_NESTED_EXCLUSIVE	Call succeeded, but this session still has nested exclusive locks.
VI_SUCCESS_NESTED_SHARED	Call succeeded, but this session still has nested shared locks.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_SESN_NLOCKED	The current session did not have any lock on the resource.

Description

This operation is used to relinquish the lock previously obtained using the `viLock()` operation.

Related Items

See the `viLock()` description in this chapter. Also see the *VISA Resource Template* description in Appendix B, *Resources*.

viUnmapAddress

Purpose

Unmaps memory space previously mapped by `viMapAddress()`.

C Syntax

```
ViStatus viUnmapAddress(ViSession vi)
```

Visual Basic Syntax

```
viUnmapAddress&(ByVal vi&)
```

Resource Classes

GPIB-VXI INSTR, GPIB-VXI MEMACC, PXI INSTR, VXI INSTR, VXI MEMACC

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.

Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_WINDOW_NMAPPED	The specified session is not currently mapped.

Description

The `viUnmapAddress()` operation unmaps the region previously mapped by the `viMapAddress()` operation for this session.

Related Items

See the `viMapAddress()` description in this chapter. Also see the *INSTR Resource* and *MEMACC Resource* descriptions in Appendix B, *Resources*.

viUnmapTrigger

Purpose

Undo a previous map from the specified trigger source line to the specified destination line.

C Syntax

```
ViStatus viUnmapTrigger(ViSession vi, ViInt16 trigSrc,
ViInt16 trigDest)
```

Visual Basic Syntax

```
viUnmapTrigger& (ByVal vi&, ByVal trigSrc%, ByVal trigDest%)
```

Resource Classes

GPIB-VXI BACKPLANE, VXI BACKPLANE

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
trigSrc	IN	Source line used in previous map. See the *Description* section for actual values.
trigDest	IN	Destination line used in previous map. See the *Description* section for actual values.

Return Values

Completion Code	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.

Error Codes	Description
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_INV_LINE	One of the specified lines (**trigSrc** or **trigDest**) is invalid.
VI_ERROR_TRIG_NMAPPED	The path from **trigSrc** to **trigDest** is not currently mapped.
VI_ERROR_NSUP_LINE	One of the specified lines (**trigSrc** or **trigDest**) is not supported by this VISA implementation.

Description

This operation can be used to undo a previous mapping of one trigger line to another. This operation is valid only on BACKPLANE (mainframe) sessions.

Special Values for trigSrc Parameters

Value	Action Description
VI_TRIG_TTL0 - VI_TRIG_TTL7	Unmap the specified VXI TTL trigger line.
VI_TRIG_ECL0 - VI_TRIG_ECL1	Unmap the specified VXI ECL trigger line.
VI_TRIG_PANEL_IN	Unmap the controller's front panel trigger input line.
VI_TRIG_PANEL_OUT	Unmap the controller's front panel trigger output line.

Special Values for trigDest Parameters

Value	Action Description
VI_TRIG_TTL0 - VI_TRIG_TTL7	Unmap the specified VXI TTL trigger line.
VI_TRIG_ECL0 - VI_TRIG_ECL1	Unmap the specified VXI ECL trigger line.
VI_TRIG_PANEL_IN	Unmap the controller's front panel trigger input line.
VI_TRIG_PANEL_OUT	Unmap the controller's front panel trigger output line.
VI_TRIG_ALL	Unmap all trigger lines to which **trigSrc** is currently connected.

This operation unmaps only one trigger mapping per call. In other words, if `viMapTrigger()` was called multiple times on the same BACKPLANE Resource and created multiple mappings for either **trigSrc** or **trigDest**, trigger mappings other than the one specified by **trigSrc** and **trigDest** should remain in effect after this call completes.

Related Items

See `viMapTrigger()` from this chapter and the *BACKPLANE Resource* description in Appendix B, *Resources*.

viVPrintf

Purpose

Converts, formats, and sends the parameters designated by **params** to the device or interface as specified by the format string.

C Syntax

```
ViStatus viVPrintf(ViSession vi, ViString writeFmt, ViVAList params)
```

Visual Basic Syntax

```
viVPrintf&(ByVal vi&, ByVal writeFmt$, params as Any)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
writeFmt	IN	String describing the format to apply to **params**.
params	IN	A list containing the variable number of parameters on which the format string is applied. The formatted data is written to the specified device.

Return Values

Completion Codes	Description
VI_SUCCESS	Parameters were successfully formatted.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.

Error Codes	Description
VI_ERROR_IO	Could not perform write operation because of I/O error.
VI_ERROR_TMO	Timeout expired before write operation completed.
VI_ERROR_INV_FMT	A format specifier in the **writeFmt** string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the **writeFmt** string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

This operation is similar to `viPrintf()`, except that the **params** parameters list provides the parameters rather than separate **arg** parameters.

Related Items

See the `viPrintf()`, `viSPrintf()`, `viVSPrintf()`, and `viVSScanf()` descriptions in this chapter. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viVQueryf

Purpose

Performs a formatted write and read through a single call to an operation.

C Syntax

```
ViStatus viVQueryf(ViSession vi, ViString writeFmt, ViString readFmt,
ViVAList params)
```

Visual Basic Syntax

```
viVQueryf&(ByVal vi&, ByVal writeFmt$, ByVal readFmt$, params as Any)
```

Resource Classes

GPIB INSTR, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
writeFmt	IN	String describing the format of write arguments.
readFmt	IN	String describing the format of read arguments.
params	IN/OUT	A list containing the variable number of write and read parameters. The write parameters are formatted and written to the specified device. The read parameters store the data read from the device after the format string is applied to the data.

Return Values

Completion Codes	Description
VI_SUCCESS	Successfully completed the query operation.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_IO	Could not perform read/write operation because of I/O error.
VI_ERROR_TMO	Timeout occurred before read/write operation completed.
VI_ERROR_INV_FMT	A format specifier in the **writeFmt** or **readFmt** string is invalid.
VI_ERROR_NSUP_FMT	The format specifier is not supported for current argument type.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

This operation is similar to `viQueryf()`, except that the **params** parameters list provides the parameters rather than the separate **arg** parameter list.

Note Because the prototype for this function cannot provide complete type-checking, remember that all output parameters must be passed by reference.

Related Items

See the `viQueryf()` description in this chapter. Also see the *INSTR Resource* and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viVScanf

Purpose

Reads, converts, and formats data using the format specifier. Stores the formatted data in the parameters designated by **params**.

C Syntax

```
ViStatus viVScanf(ViSession vi, ViString readFmt, ViVAList params)
```

Visual Basic Syntax

```
viVScanf&(ByVal vi&, ByVal readFmt$, params as Any)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
readFmt	IN	String describing the format to apply to **params**.
params	OUT	A list with the variable number of parameters into which the data is read and the format string is applied.

Return Values

Completion Codes	Description
VI_SUCCESS	Data was successfully read and formatted into **params** parameter(s).

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.

Error Codes	Description
VI_ERROR_IO	Could not perform read operation because of I/O error.
VI_ERROR_TMO	Timeout expired before read operation completed.
VI_ERROR_INV_FMT	A format specifier in the **readFmt** string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the **readFmt** string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

This operation is similar to viScanf(), except that the **params** parameters list provides the parameters rather than separate **arg** parameters.

Note Because the prototype for this function cannot provide complete type-checking, remember that all output parameters must be passed by reference.

Related Items

See the viScanf(), viSScanf(), viVPrintf(), and viVSScanf() descriptions in this chapter. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viVSPrintf

Purpose

Converts, formats, and sends the parameters designated by **params** to a user-specified buffer as specified by the format string.

C Syntax

```
ViStatus viVSPrintf(ViSession vi, ViPBuf buf, ViString writeFmt,
ViVAList params)
```

Visual Basic Syntax

```
viVSPrintf&(ByVal vi&, ByVal buf$, ByVal writeFmt$, params as Any)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	OUT	Buffer where data is to be written.
writeFmt	IN	The format string to apply to parameters in `ViVAList`.
params	IN	A list containing the variable number of parameters on which the format string is applied. The formatted data is written to the specified **buf**.

Return Values

Completion Codes	Description
`VI_SUCCESS`	Parameters were successfully formatted.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_INV_FMT	A format specifier in the **writeFmt** string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the **writeFmt** string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

This operation is similar to `viVPrintf()`, except that the output is not written to the device; it is written to the user-specified buffer. This output buffer is NULL terminated.

If this operation outputs an END indicator before all the arguments are satisfied, then the rest of the **writeFmt** string is ignored and the buffer string is still terminated by a NULL.

Note The size of the **buf** parameter should be large enough to hold the formatted I/O contents plus the NULL termination character.

Related Items

See the `viPrintf()`, `viSPrintf()`, `viVPrintf()`, and `viVSScanf()` descriptions in this chapter. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viVSScanf

Purpose

Reads, converts, and formats data from a user-specified buffer using the format specifier. Stores the formatted data in the parameters designated by **params**.

C Syntax

```
ViStatus viVSScanf(ViSession vi, ViBuf buf, ViString readFmt,
ViVAList params)
```

Visual Basic Syntax

```
viVSScanf&(ByVal vi&, ByVal buf$, ByVal readFmt$, params as Any)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	IN	Buffer from which data is read and formatted.
readFmt	IN	String describing the format to apply to **params**.
params	OUT	A list with the variable number of parameters into which the data is read and the format string is applied.

Return Values

Completion Codes	Description
VI_SUCCESS	Data was successfully read and formatted into **params** parameter(s).

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_INV_FMT	A format specifier in the **readFmt** string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the **readFmt** string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

Description

The viVSScanf() operation is similar to viVScanf(), except that the data is read from a user-specified buffer rather than a device.

Note Because the prototype for this function cannot provide complete type checking, remember that all output parameters must be passed by reference.

Related Items

See the viScanf(), viSScanf(), viVSPrintf(), and viVScanf() descriptions in this chapter. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viVxiCommandQuery

Purpose

Sends the device a miscellaneous command or query and/or retrieves the response to a previous query.

C Syntax

```
ViStatus viVxiCommandQuery(ViSession vi, ViUInt16 mode,
ViUInt32 cmd, ViPUInt32 response)
```

Visual Basic Syntax

```
viVxiCommandQuery&(ByVal vi&, ByVal mode%, ByVal cmd&, response&)
```

Resource Classes

GPIB-VXI INSTR, VXI INSTR

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
mode	IN	Specifies whether to issue a command and/or retrieve a response. See the *Description* section for actual values.
cmd	IN	The miscellaneous command to send.
response	OUT	The response retrieved from the device. If the mode specifies to send a command rather than retrieve a response, you can use `VI_NULL` for this parameter.

Return Values

Completion Codes	Description
`VI_SUCCESS`	The operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_OUTP_PROT_VIOL	Device reported an output protocol error during transfer.
VI_ERROR_INP_PROT_VIOL	Device reported an input protocol error during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_RESP_PENDING	A previous response is still pending, causing a multiple query error.
VI_ERROR_INV_MODE	The value specified by the **mode** parameter is invalid.

Description

The `viVxiCommandQuery()` operation can send a command or query, or receive a response to a query previously sent to the device. The **mode** parameter specifies whether to issue a command and/or retrieve a response, and indicates the type or size of command and/or response to use. The following table defines the values for the **mode** parameter.

Mode	Action Description
VI_VXI_CMD16	Send 16-bit Word Serial command.
VI_VXI_CMD16_RESP16	Send 16-bit Word Serial query; get 16-bit response.
VI_VXI_RESP16	Get 16-bit response from previous query.
VI_VXI_CMD32	Send 32-bit Word Serial command.
VI_VXI_CMD32_RESP16	Send 32-bit Word Serial query; get 16-bit response.

Mode	Action Description
VI_VXI_CMD32_RESP32	Send 32-bit Word Serial query; get 32-bit response.
VI_VXI_RESP32	Get 32-bit response from previous query.

Notice that the **mode** you specify can cause all or part of the **cmd** or **response** parameters to be ignored.

- If **mode** specifies sending a 16-bit command, the upper half of **cmd** is ignored.
- If **mode** specifies retrieving a response only, **cmd** is ignored.
- If **mode** specifies sending a command only, **response** is ignored. You can use VI_NULL for the value of **response**.
- If **mode** specifies to retrieve a 16-bit value, the upper half of **response** is set to 0.

Related Items

See the *INSTR Resource* description in Appendix B, *Resources*. Also refer to the *VXI Specification* for defined Word Serial commands.

viWaitOnEvent

Purpose

Waits for an occurrence of the specified event for a given session.

C Syntax

```
ViStatus viWaitOnEvent(ViSession vi, ViEventType inEventType,
ViUInt32 timeout, ViPEventType outEventType, ViPEvent outContext)
```

Visual Basic Syntax

```
viWaitOnEvent&(ByVal vi&, ByVal inEventType&, ByVal timeout&,
outEventType&, outContext&)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, GPIB-VXI MEMACC, GPIB-VXI BACKPLANE, PXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
inEventType	IN	Logical identifier of the event(s) to wait for.
timeout	IN	Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds.
outEventType	OUT	Logical identifier of the event actually received.
outContext	OUT	A handle specifying the unique occurrence of an event.

Return Values

Completion Codes	Description
VI_SUCCESS	Wait terminated successfully on receipt of an event occurrence. The queue is empty.
VI_SUCCESS_QUEUE_NEMPTY	Wait terminated successfully on receipt of an event notification. There is still at least one more event occurrence of the type specified by **inEventType** available for this session.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_INV_EVENT	Specified event type is not supported by the resource.
VI_ERROR_TMO	Specified event did not occur within the specified time period.
VI_ERROR_NENABLED	The session must be enabled for events of the specified type in order to receive them.

Description

The `viWaitOnEvent()` operation suspends the execution of a thread of an application and waits for an event of the type specified by **inEventType** for a time period specified by **timeout**. You can wait only for events that have been enabled with the `viEnableEvent()` operation. Refer to individual event descriptions for context definitions. If the specified **inEventType** is `VI_ALL_ENABLED_EVENTS`, the operation waits for any event that is enabled for the given session. If the specified timeout value is `VI_TMO_INFINITE`, the operation is suspended indefinitely. If the specified timeout value is `VI_TMO_IMMEDIATE`, the operation is not suspended; therefore, this value can be used to dequeue events from an event queue.

When the **outContext** handle returned from a successful invocation of `viWaitOnEvent()` is no longer needed, it should be passed to `viClose()`.

If a session's event queue becomes full and a new event arrives, the new event is discarded. The default event queue size (per session) is 50, which is sufficiently large for most applications. If an application expects more than 50 events to arrive without having been handled, it can modify the value of the attribute `VI_ATTR_MAX_QUEUE_LENGTH` to the required size.

The **outEventType** and **outContext** parameters are optional and can be VI_NULL. This can be used if the event type is known from the **inEventType** parameter, or if the **outContext** handle is not needed to retrieve additional information. If VI_NULL is used for the **outContext** parameter, VISA will automatically close the event context.

Related Items

See the viEnableEvent() and viClose() descriptions in this chapter. See the VI_ATTR_MAX_QUEUE_LENGTH description in Chapter 3, *Attributes*. See Chapter 4, *Events*, for a list of events that you can wait for. Also see the *VISA Resource Template*, description in Appendix B, *Resources*.

viWrite

Purpose

Writes data to device or interface synchronously.

C Syntax

```
ViStatus viWrite(ViSession vi, ViBuf buf, ViUInt32 count,
ViPUInt32 retCount)
```

Visual Basic Syntax

```
viWrite&(ByVal vi&, ByVal buf$, ByVal count&, retCount&)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	IN	Location of a data block to be sent to a device.
count	IN	Number of bytes to be written.
retCount	OUT	Number of bytes actually transferred.

Return Values

Completion Codes	Description
VI_SUCCESS	Transfer completed.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.

Error Codes	Description
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_INP_PROT_VIOL	Device reported an input protocol error during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SETUP	Unable to start write operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_NCIC	The interface associated with the given **vi** is not currently the controller in charge.
VI_ERROR_NLISTENERS	No-listeners condition is detected (both NRFD and NDAC are unasserted).
VI_ERROR_IO	An unknown I/O error occurred during transfer.
VI_ERROR_CONN_LOST	The I/O connection for the given session has been lost.

Description

The viWrite() operation synchronously transfers data. The data to be written is in the buffer represented by **buf**. This operation returns only when the transfer terminates. Only one synchronous write operation can occur at any one time.

Related Items

See the viRead(), viBufWrite(), viWriteAsync(), and viWriteFromFile() descriptions in this chapter. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viWriteAsync

Purpose

Writes data to device or interface asynchronously.

C Syntax

```
ViStatus viWriteAsync(ViSession vi, ViBuf buf, ViUInt32 count,
ViPJobId jobId)
```

Visual Basic Syntax

N/A

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	IN	Location of a data block to be sent to a device.
count	IN	Number of bytes to be written.
jobId	OUT	Job ID of this asynchronous write operation.

Return Values

Completion Codes	Description
VI_SUCCESS	Asynchronous write operation successfully queued.
VI_SUCCESS_SYNC	Write operation performed synchronously.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_QUEUE_ERROR	Unable to queue write operation.
VI_ERROR_IN_PROGRESS	Unable to queue the asynchronous operation because there is already an operation in progress.

Description

The viWriteAsync() operation asynchronously transfers data. The data to be written is in the buffer represented by **buf**. This operation normally returns before the transfer terminates.

Before calling this operation, you should enable the session for receiving I/O completion events. After the transfer has completed, an I/O completion event is posted.

The operation returns a job identifier that you can use with either viTerminate() to abort the operation or with an I/O completion event to identify which asynchronous write operation completed. VISA will never return VI_NULL for a valid **jobId**.

Related Items

See the viEnableEvent(), viWrite(), viTerminate(), and viReadAsync() descriptions in this chapter. See the VI_EVENT_IO_COMPLETION description in Chapter 4, *Events*. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

viWriteFromFile

Purpose

Take data from a file and write it out synchronously.

C Syntax

```
viStatus viWriteFromFile(ViSession vi, ViString fileName, ViUInt32
count, ViPUInt32 retCount)
```

Visual Basic Syntax

```
viWriteFrom File& (ByVal vi&, ByVal filename$, ByVal count&,
retCount&)
```

Resource Classes

GPIB INSTR, GPIB INTFC, GPIB SERVANT, GPIB-VXI INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
fileName	IN	Name of file from which data will be read.
count	IN	Number of bytes to be written.
retCount	OUT	Number of bytes actually transferred.

Return Values

Completion Code	Description
VI_SUCCESS	Transfer completed.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_OPER	The given **vi** does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by **vi** has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_RAW_WR_PROT_VIOL	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	Violation of raw read protocol occurred during transfer.
VI_ERROR_INP_PROT_VIOL	Device reported an input protocol error during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_NCIC	The interface associated with the given **vi** is not currently the controller in charge.
VI_ERROR_NLISTENERS	No Listeners condition is detected (both NRFD and NDAC are deasserted).
VI_ERROR_IO	An unknown I/O error occurred during transfer.
VI_ERROR_FILE_ACCESS	An error occurred while trying to open the specified file. Possible reasons include an invalid path or lack of access rights.

Error Codes	Description
VI_ERROR_FILE_IO	An error occurred while accessing the specified file.
VI_ERROR_CONN_LOST	The I/O connection for the given session has been lost.

Description

This write operation synchronously transfers data. The file specified in **fileName** is opened in binary read-only mode, and the data (up to end-of-file or the number of bytes specified in count) is read. The data is then written to the device. This operation returns only when the transfer terminates.

This operation is useful for sending data that was already processed and/or formatted.

Special Values for retCount Parameter

Value	Action Description
VI_NULL	Do not return the number of bytes transferred.

If you pass VI_NULL as the **retCount** parameter to the viWriteFromFile() operation, the number of bytes transferred will not be returned. This may be useful if it is important to know only whether the operation succeeded or failed.

Related Items

See viWrite() and viReadToFile() in this chapter. Also see the *INSTR Resource*, *INTFC Resource*, *SERVANT Resource*, and *SOCKET Resource* descriptions in Appendix B, *Resources*.

A

Status Codes

This appendix lists and describes the completion and error codes.

Table A-1. Completion Codes

Completion Codes	Values	Meaning
VI_SUCCESS	0	Operation completed successfully.
VI_SUCCESS_EVENT_EN	3FFF0002h	Specified event is already enabled for at least one of the specified mechanisms.
VI_SUCCESS_EVENT_DIS	3FFF0003h	Specified event is already disabled for at least one of the specified mechanisms.
VI_SUCCESS_QUEUE_EMPTY	3FFF0004h	Operation completed successfully, but queue was already empty.
VI_SUCCESS_TERM_CHAR	3FFF0005h	The specified termination character was read.
VI_SUCCESS_MAX_CNT	3FFF0006h	The number of bytes read is equal to the input count.
VI_WARN_CONFIG_NLOADED	3FFF0077h	The specified configuration either does not exist or could not be loaded; using VISA-specified defaults.
VI_SUCCESS_DEV_NPRESENT	3FFF007Dh	Session opened successfully, but the device at the specified address is not responding.
VI_SUCCESS_TRIG_MAPPED	3FFF007Eh	The path from trigSrc to trigDest is already mapped.
VI_SUCCESS_QUEUE_NEMPTY	3FFF0080h	Wait terminated successfully on receipt of an event notification. There is still at least one more event occurrence of the requested type(s) available for this session.

Table A-1. Completion Codes (Continued)

Completion Codes	Values	Meaning
VI_WARN_NULL_OBJECT	3FFF0082h	The specified object reference is uninitialized.
VI_WARN_NSUP_ATTR_STATE	3FFF0084h	Although the specified state of the attribute is valid, it is not supported by this resource implementation.
VI_WARN_UNKNOWN_STATUS	3FFF0085h	The status code passed to the operation could not be interpreted.
VI_WARN_NSUP_BUF	3FFF0088h	The specified buffer is not supported.
VI_SUCCESS_NCHAIN	3FFF0098h	Event handled successfully. Do not invoke any other handlers on this session for this event.
VI_SUCCESS_NESTED_SHARED	3FFF0099h	Operation completed successfully, and this session has nested shared locks.
VI_SUCCESS_NESTED_EXCLUSIVE	3FFF009Ah	Operation completed successfully, and this session has nested exclusive locks.
VI_SUCCESS_SYNC	3FFF009Bh	Asynchronous operation request was actually performed synchronously.

Table A-2. Error Codes

Error Codes	Values	Meaning
VI_ERROR_SYSTEM_ERROR	BFFF0000h	Unknown system error (miscellaneous error).
VI_ERROR_INV_OBJECT	BFFF000Eh	The given session or object reference is invalid.
VI_ERROR_RSRC_LOCKED	BFFF000Fh	Specified type of lock cannot be obtained or specified operation cannot be performed, because the resource is locked.
VI_ERROR_INV_EXPR	BFFF0010h	Invalid expression specified for search.
VI_ERROR_RSRC_NFOUND	BFFF0011h	Insufficient location information or the device or resource is not present in the system.

Table A-2. Error Codes (Continued)

Error Codes	Values	Meaning
VI_ERROR_INV_RSRC_NAME	BFFF0012h	Invalid resource reference specified. Parsing error.
VI_ERROR_INV_ACC_MODE	BFFF0013h	Invalid access mode.
VI_ERROR_TMO	BFFF0015h	Timeout expired before operation completed.
VI_ERROR_CLOSING_FAILED	BFFF0016h	Unable to deallocate the previously allocated data structures corresponding to this session or object reference.
VI_ERROR_INV_DEGREE	BFFF001Bh	Specified degree is invalid.
VI_ERROR_INV_JOB_ID	BFFF001Ch	Specified job identifier is invalid.
VI_ERROR_NSUP_ATTR	BFFF001Dh	The specified attribute is not defined or supported by the referenced session, event, or find list.
VI_ERROR_NSUP_ATTR_STATE	BFFF001Eh	The specified state of the attribute is not valid, or is not supported as defined by the session, event, or find list.
VI_ERROR_ATTR_READONLY	BFFF001Fh	The specified attribute is read-only.
VI_ERROR_INV_LOCK_TYPE	BFFF0020h	The specified type of lock is not supported by this resource.
VI_ERROR_INV_ACCESS_KEY	BFFF0021h	The access key to the resource associated with this session is invalid.
VI_ERROR_INV_EVENT	BFFF0026h	Specified event type is not supported by the resource.
VI_ERROR_INV_MECH	BFFF0027h	Invalid mechanism specified.
VI_ERROR_HNDLR_NINSTALLED	BFFF0028h	A handler is not currently installed for the specified event.
VI_ERROR_INV_HNDLR_REF	BFFF0029h	The given handler reference is invalid.
VI_ERROR_INV_CONTEXT	BFFF002Ah	Specified event context is invalid.
VI_ERROR_QUEUE_OVERFLOW	BFFF002Dh	The event queue for the specified type has overflowed (usually due to previous events not having been closed).

Table A-2. Error Codes (Continued)

Error Codes	Values	Meaning
VI_ERROR_NENABLED	BFFF002Fh	The session must be enabled for events of the specified type in order to receive them.
VI_ERROR_ABORT	BFFF0030h	The operation was aborted.
VI_ERROR_RAW_WR_PROT_VIOL	BFFF0034h	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	BFFF0035h	Violation of raw read protocol occurred during transfer.
VI_ERROR_OUTP_PROT_VIOL	BFFF0036h	Device reported an output protocol error during transfer.
VI_ERROR_INP_PROT_VIOL	BFFF0037h	Device reported an input protocol error during transfer.
VI_ERROR_BERR	BFFF0038h	Bus error occurred during transfer.
VI_ERROR_IN_PROGRESS	BFFF0039h	Unable to queue the asynchronous operation because there is already an operation in progress.
VI_ERROR_INV_SETUP	BFFF003Ah	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_QUEUE_ERROR	BFFF003Bh	Unable to queue asynchronous operation.
VI_ERROR_ALLOC	BFFF003Ch	Insufficient system resources to perform necessary memory allocation.
VI_ERROR_INV_MASK	BFFF003Dh	Invalid buffer mask specified.
VI_ERROR_IO	BFFF003Eh	Could not perform operation because of I/O error.
VI_ERROR_INV_FMT	BFFF003Fh	A format specifier in the format string is invalid.
VI_ERROR_NSUP_FMT	BFFF0041h	A format specifier in the format string is not supported.
VI_ERROR_LINE_IN_USE	BFFF0042h	The specified trigger line is currently in use.

Table A-2. Error Codes (Continued)

Error Codes	Values	Meaning
VI_ERROR_NSUP_MODE	BFFF0046h	The specified mode is not supported by this VISA implementation.
VI_ERROR_SRQ_NOCCURRED	BFFF004Ah	Service request has not been received for the session.
VI_ERROR_INV_SPACE	BFFF004Eh	Invalid address space specified.
VI_ERROR_INV_OFFSET	BFFF0051h	Invalid offset specified.
VI_ERROR_INV_WIDTH	BFFF0052h	Invalid source or destination width specified.
VI_ERROR_NSUP_OFFSET	BFFF0054h	Specified offset is not accessible from this hardware.
VI_ERROR_NSUP_VAR_WIDTH	BFFF0055h	Cannot support source and destination widths that are different.
VI_ERROR_WINDOW_NMAPPED	BFFF0057h	The specified session is not currently mapped.
VI_ERROR_RESP_PENDING	BFFF0059h	A previous response is still pending, causing a multiple query error.
VI_ERROR_NLISTENERS	BFFF005Fh	No Listeners condition is detected (both NRFD and NDAC are deasserted).
VI_ERROR_NCIC	BFFF0060h	The interface associated with this session is not currently the controller in charge.
VI_ERROR_NSYS_CNTLR	BFFF0061h	The interface associated with this session is not the system controller.
VI_ERROR_NSUP_OPER	BFFF0067h	The given session or object reference does not support this operation.
VI_ERROR_INTR_PENDING	BFFF0068h	An interrupt is still pending from a previous call.
VI_ERROR_ASRL_PARITY	BFFF006Ah	A parity error occurred during transfer.
VI_ERROR_ASRL_FRAMING	BFFF006Bh	A framing error occurred during transfer.

Table A-2. Error Codes (Continued)

Error Codes	Values	Meaning
VI_ERROR_ASRL_OVERRUN	BFFF006Ch	An overrun error occurred during transfer. A character was not read from the hardware before the next character arrived.
VI_ERROR_TRIG_NMAPPED	BFFF006Eh	The path from trigSrc to trigDest is not currently mapped.
VI_ERROR_NSUP_ALIGN_OFFSET	BFFF0070h	The specified offset is not properly aligned for the access width of the operation.
VI_ERROR_USER_BUF	BFFF0071h	A specified user buffer is not valid or cannot be accessed for the required size.
VI_ERROR_RSRC_BUSY	BFFF0072h	The resource is valid, but VISA cannot currently access it.
VI_ERROR_NSUP_WIDTH	BFFF0076h	Specified width is not supported by this hardware.
VI_ERROR_INV_PARAMETER	BFFF0078h	The value of some parameter—which parameter is not known—is invalid.
VI_ERROR_INV_PROT	BFFF0079h	The protocol specified is invalid.
VI_ERROR_INV_SIZE	BFFF007Bh	Invalid size of window specified.
VI_ERROR_WINDOW_MAPPED	BFFF0080h	The specified session currently contains a mapped window.
VI_ERROR_NIMPL_OPER	BFFF0081h	The given operation is not implemented.
VI_ERROR_INV_LENGTH	BFFF0083h	Invalid length specified.
VI_ERROR_INV_MODE	BFFF0091h	The specified mode is invalid.
VI_ERROR_SESN_NLOCKED	BFFF009Ch	The current session did not have any lock on the resource.
VI_ERROR_MEM_NSHARED	BFFF009Dh	The device does not export any memory.
VI_ERROR_LIBRARY_NFOUND	BFFF009Eh	A code library required by VISA could not be located or loaded.

Table A-2. Error Codes (Continued)

Error Codes	Values	Meaning
VI_ERROR_NSUP_INTR	BFFF009Fh	The interface cannot generate an interrupt on the requested level or with the requested statusID value.
VI_ERROR_INV_LINE	BFFF00A0h	The value specified by the line parameter is invalid.
VI_ERROR_FILE_ACCESS	BFFF00A1h	An error occurred while trying to open the specified file. Possible reasons include an invalid path or lack of access rights.
VI_ERROR_FILE_IO	BFFF00A2h	An error occurred while performing I/O on the specified file.
VI_ERROR_NSUP_LINE	BFFF00A3h	One of the specified lines (trigSrc or trigDest) is not supported by this VISA implementation, or the combination of lines is not a valid mapping.
VI_ERROR_NSUP_MECH	BFFF00A4h	The specified mechanism is not supported by the given event type.
VI_ERROR_INTF_NUM_NCONFIG	BFFF00A5h	The interface type is valid but the specified interface number is not configured.
VI_ERROR_CONN_LOST	BFFF00A6h	The connection for the given session has been lost.

B

Resources

This appendix lists the attributes, events, and operations in each resource in VISA. Refer to Chapter 3, *Attributes*, Chapter 4, *Events*, and Chapter 5, *Operations*, for more details.

VISA Resource Template

This section lists the attributes, events, and operations for the VISA Resource Template. The attributes, events, and operations in the VISA Resource Template are available to all other resources.

Attributes

VI_ATTR_MAX_QUEUE_LENGTH
VI_ATTR_RM_SESSION
VI_ATTR_RSRC_CLASS
VI_ATTR_RSRC_IMPL_VERSION
VI_ATTR_RSRC_LOCK_STATE
VI_ATTR_RSRC_MANF_ID
VI_ATTR_RSRC_MANF_NAME
VI_ATTR_RSRC_NAME
VI_ATTR_RSRC_SPEC_VERSION
VI_ATTR_USER_DATA

Events

VI_EVENT_EXCEPTION

Operations

```
viClose(vi)
viDisableEvent(vi, eventType, mechanism)
viDiscardEvents(vi, eventType, mechanism)
viEnableEvent(vi, eventType, mechanism, context)
viGetAttribute(vi, attribute, attrState)
viInstallHandler(vi, eventType, handler,userHandle)
viLock(vi, lockType, timeout, requestedKey,
       accessKey)
viSetAttribute(vi, attribute, attrState)
```

```
viStatusDesc(vi, status, desc)
viTerminate(vi, degree, jobId)
viUninstallHandler(vi, eventType, handler,
                   userHandle)
viUnlock(vi)
viWaitOnEvent(vi, inEventType, timeout,
              outEventType, outContext)
```

VISA Resource Manager

This section lists the attributes, events, and operations for the VISA Resource Manager. The attributes, events, and operations in the VISA Resource Template are available to this resource in addition to the operations listed below.

Attributes

The attributes for the VISA Resource Template are available to this resource. This resource has no defined attributes of its own.

Events

None

Operations

```
viFindNext(findList, instrDesc)
viFindRsrc(sesn, expr, findList, retcnt,instrDesc)
viOpen(sesn, rsrcName, accessMode, timeout, vi)
viOpenDefaultRM(sesn)
viParseRsrc(sesn, rsrcName, intfType, intfNum)
```

INSTR Resource

This section lists the attributes, events, and operations for the INSTR Resource. The attributes, events, and operations in the VISA Resource Template are available to this resource in addition to the attributes and operations listed below.

Attributes

```
VI_ATTR_ASRL_AVAIL_NUM
VI_ATTR_ASRL_BAUD
VI_ATTR_ASRL_CTS_STATE
VI_ATTR_ASRL_DATA_BITS
```

VI_ATTR_ASRL_DCD_STATE
VI_ATTR_ASRL_DSR_STATE
VI_ATTR_ASRL_DTR_STATE
VI_ATTR_ASRL_END_IN
VI_ATTR_ASRL_END_OUT
VI_ATTR_ASRL_FLOW_CNTRL
VI_ATTR_ASRL_PARITY
VI_ATTR_ASRL_REPLACE_CHAR
VI_ATTR_ASRL_RI_STATE
VI_ATTR_ASRL_RTS_STATE
VI_ATTR_ASRL_STOP_BITS
VI_ATTR_ASRL_XOFF_CHAR
VI_ATTR_ASRL_XON_CHAR
VI_ATTR_CMDR_LA
VI_ATTR_DEST_ACCESS_PRIV
VI_ATTR_DEST_BYTE_ORDER
VI_ATTR_DEST_INCREMENT
VI_ATTR_DMA_ALLOW_EN
VI_ATTR_FDC_CHNL
VI_ATTR_FDC_MODE
VI_ATTR_FDC_USE_PAIR
VI_ATTR_FILE_APPEND_EN
VI_ATTR_GPIB_PRIMARY_ADDR
VI_ATTR_GPIB_READDR_EN
VI_ATTR_GPIB_REN_STATE
VI_ATTR_GPIB_SECONDARY_ADDR
VI_ATTR_GPIB_UNADDR_EN
VI_ATTR_IMMEDIATE_SERV
VI_ATTR_INTF_INST_NAME
VI_ATTR_INTF_NUM
VI_ATTR_INTF_PARENT_NUM
VI_ATTR_INTF_TYPE
VI_ATTR_IO_PROT
VI_ATTR_MAINFRAME_LA
VI_ATTR_MANF_ID
VI_ATTR_MANF_NAME
VI_ATTR_MEM_BASE
VI_ATTR_MEM_SIZE
VI_ATTR_MEM_SPACE
VI_ATTR_MODEL_CODE
VI_ATTR_MODEL_NAME
VI_ATTR_RD_BUF_OPER_MODE
VI_ATTR_SEND_END_EN
VI_ATTR_SLOT

VI_ATTR_SRC_ACCESS_PRIV
VI_ATTR_SRC_BYTE_ORDER
VI_ATTR_SRC_INCREMENT
VI_ATTR_SUPPRESS_END_EN
VI_ATTR_TCPIP_ADDR
VI_ATTR_TCPIP_DEVICE_NAME
VI_ATTR_TCPIP_HOSTNAME
VI_ATTR_TERMCHAR
VI_ATTR_TERMCHAR_EN
VI_ATTR_TMO_VALUE
VI_ATTR_TRIG_ID
VI_ATTR_VXI_DEV_CLASS
VI_ATTR_VXI_LA
VI_ATTR_VXI_TRIG_SUPPORT
VI_ATTR_WIN_ACCESS
VI_ATTR_WIN_ACCESS_PRIV
VI_ATTR_WIN_BASE_ADDR
VI_ATTR_WIN_BYTE_ORDER
VI_ATTR_WIN_SIZE
VI_ATTR_WR_BUF_OPER_MODE

Events

VI_EVENT_IO_COMPLETION
VI_EVENT_SERVICE_REQ
VI_EVENT_TRIG
VI_EVENT_VXI_SIGP
VI_EVENT_VXI_VME_INTR

Operations

viAssertTrigger(vi, protocol)
viBufRead(vi, buf, count, retCount)
viBufWrite(vi, buf, count, retCount)
viClear(vi)
viFlush(vi, mask)
viGpibControlREN(vi, mode)
viIn8(vi, space, offset, val8)
viIn16(vi, space, offset, val16)
viIn32(vi, space, offset, val32)
viMapAddress(vi, mapSpace, mapBase, mapSize, access,
suggested, address)
viMemAlloc(vi, size, offset)
viMemFree(vi, offset)

```
viMove(vi, srcSpace, srcOffset, srcWidth, destSpace,
       destOffset, destWidth, length)
viMoveAsync(vi, srcSpace, srcOffset, srcWidth,
            destSpace, destOffset, destWidth,
            length, jobId)
viMoveIn8(vi, space, offset, length, buf8)
viMoveIn16(vi, space, offset, length, buf16)
viMoveIn32(vi, space, offset, length, buf32)
viMoveOut8(vi, space, offset, length, buf8)
viMoveOut16(vi, space, offset, length, buf16)
viMoveOut32(vi, space, offset, length, buf32)
viOut8(vi, space, offset, val8)
viOut16(vi, space, offset, val16)
viOut32(vi, space, offset, val32)
viPeek8(vi, addr, val8)
viPeek16(vi, addr, val16)
viPeek32(vi, addr, val32)
viPoke8(vi, addr, val8)
viPoke16(vi, addr, val16)
viPoke32(vi, addr, val32)
viPrintf(vi, writeFmt, ...)
viQueryf(vi, writeFmt, readFmt, ...)
viRead(vi, buf, count, retCount)
viReadAsync(vi, buf, count, jobId)
viReadSTB(vi, status)
viReadToFile(vi, fileName, count, retCount)
viScanf(vi, readFmt, ...)
viSetBuf(vi, mask, size)
viSPrintf(vi, buf, writeFmt, ...)
viSScanf(vi, buf, readFmt, ...)
viUnmapAddress(vi)
viVPrintf(vi, writeFmt, params)
viVQueryf(vi, writeFmt, readFmt, params)
viVScanf(vi, readFmt, params)
viVSPrintf(vi, buf, writeFmt, params)
viVSScanf(vi, buf, readFmt, params)
viVxiCommandQuery(vi, mode, cmd, response)
viWrite(vi, buf, count, retCount)
viWriteAsync(vi, buf, count, jobId)
viWriteFromFile(vi, fileName, count, retCount)
```

MEMACC Resource

This section lists the attributes, events, and operations for the MEMACC Resource. The attributes, events, and operations in the VISA Resource Template are available to this resource in addition to the attributes and operations listed below.

Attributes

VI_ATTR_DEST_ACCESS_PRIV
VI_ATTR_DEST_BYTE_ORDER
VI_ATTR_DEST_INCREMENT
VI_ATTR_DMA_ALLOW_EN
VI_ATTR_GPIB_PRIMARY_ADDR
VI_ATTR_GPIB_SECONDARY_ADDR
VI_ATTR_INTF_INST_NAME
VI_ATTR_INTF_NUM
VI_ATTR_INTF_PARENT_NUM
VI_ATTR_INTF_TYPE
VI_ATTR_SRC_ACCESS_PRIV
VI_ATTR_SRC_BYTE_ORDER
VI_ATTR_SRC_INCREMENT
VI_ATTR_TMO_VALUE
VI_ATTR_VXI_LA
VI_ATTR_WIN_ACCESS
VI_ATTR_WIN_ACCESS_PRIV
VI_ATTR_WIN_BASE_ADDR
VI_ATTR_WIN_BYTE_ORDER
VI_ATTR_WIN_SIZE

Events

VI_EVENT_IO_COMPLETION

Operations

```
viIn8(vi, space, offset, val8)
viIn16(vi, space, offset, val16)
viIn32(vi, space, offset, val32)
viMapAddress(vi, mapSpace, mapBase, mapSize, access,
             suggested, address)
viMove(vi, srcSpace, srcOffset, srcWidth, destSpace,
        destOffset, destWidth, length)
```

```
viMoveAsync(vi, srcSpace, srcOffset, srcWidth,
            destSpace, destOffset, destWidth,
            length, jobId)
viMoveIn8(vi, space, offset, length, buf8)
viMoveIn16(vi, space, offset, length, buf16)
viMoveIn32(vi, space, offset, length, buf32)
viMoveOut8(vi, space, offset, length, buf8)
viMoveOut16(vi, space, offset, length, buf16)
viMoveOut32(vi, space, offset, length, buf32)
viOut8(vi, space, offset, val8)
viOut16(vi, space, offset, val16)
viOut32(vi, space, offset, val32)
viPeek8(vi, addr, val8)
viPeek16(vi, addr, val16)
viPeek32(vi, addr, val32)
viPoke8(vi, addr, val8)
viPoke16(vi, addr, val16)
viPoke32(vi, addr, val32)
viUnmapAddress(vi)
```

INTFC Resource

This section lists the attributes, events, and operations for the INTFC Resource. The attributes, events, and operations in the VISA Resource Template are available to this resource in addition to the attributes and operations listed below.

Attributes

VI_ATTR_DEV_STATUS_BYTE
VI_ATTR_DMA_ALLOW_EN
VI_ATTR_FILE_APPEND_EN
VI_ATTR_GPIB_ATN_STATE
VI_ATTR_GPIB_CIC_STATE
VI_ATTR_GPIB_HS488_CBL_LEN
VI_ATTR_GPIB_NDAC_STATE
VI_ATTR_GPIB_PRIMARY_ADDR
VI_ATTR_GPIB_REN_STATE
VI_ATTR_GPIB_SECONDARY_ADDR
VI_ATTR_GPIB_SRQ_STATE
VI_ATTR_GPIB_SYS_CNTRL_STATE
VI_ATTR_INTF_INST_NAME
VI_ATTR_INTF_NUM
VI_ATTR_INTF_TYPE

VI_ATTR_RD_BUF_OPER_MODE
VI_ATTR_SEND_END_EN
VI_ATTR_TERMCHAR
VI_ATTR_TERMCHAR_EN
VI_ATTR_TMO_VALUE
VI_ATTR_TRIG_ID
VI_ATTR_WR_BUF_OPER_MODE

Events

VI_EVENT_CLEAR
VI_EVENT_GPIB_CIC
VI_EVENT_GPIB_LISTEN
VI_EVENT_GPIB_TALK
VI_EVENT_IO_COMPLETION
VI_EVENT_SERVICE_REQ
VI_EVENT_TRIG

Operations

viAssertTrigger(vi, protocol)
viBufRead(vi, buf, count, retCount)
viBufWrite(vi, buf, count, retCount)
viFlush(vi, mask)
viGpibCommand(vi, buf, count, retCount)
viGpibControlATN(vi, mode)
viGpibControlREN(vi, mode)
viGpibPassControl(vi, primAddr, secAddr)
viGpibSendIFC(vi)
viPrintf(vi, writeFmt, ...)
viRead(vi, buf, count, retCount)
viReadAsync(vi, buf, count, jobId)
viReadToFile(vi, fileName, count, retCount)
viScanf(vi, readFmt, ...)
viSetBuf(vi, mask, size)
viSPrintf(vi, buf, writeFmt, ...)
viSScanf(vi, buf, readFmt, ...)
viVPrintf(vi, writeFmt, params)
viVScanf(vi, readFmt, params)
viVSPrintf(vi, buf, writeFmt, params)
viVSScanf(vi, buf, readFmt, params)
viWrite(vi, buf, count, retCount)
viWriteAsync(vi, buf, count, jobId)
viWriteFromFile(vi, fileName, count, retCount)

BACKPLANE Resource

This section lists the attributes, events, and operations for the BACKPLANE Resource. The attributes, events, and operations in the VISA Resource Template are available to this resource in addition to the attributes and operations listed below.

Attributes

VI_ATTR_GPIB_PRIMARY_ADDR
VI_ATTR_GPIB_SECONDARY_ADDR
VI_ATTR_INTF_INST_NAME
VI_ATTR_INTF_NUM
VI_ATTR_INTF_PARENT_NUM
VI_ATTR_INTF_TYPE
VI_ATTR_MAINFRAME_LA
VI_ATTR_TMO_VALUE
VI_ATTR_TRIG_ID
VI_ATTR_VXI_TRIG_STATUS
VI_ATTR_VXI_TRIG_SUPPORT
VI_ATTR_VXI_VME_INTR_STATUS
VI_ATTR_VXI_VME_SYSFAIL_STATE

Events

VI_EVENT_TRIG
VI_EVENT_VXI_VME_SYSFAIL
VI_EVENT_VXI_VME_SYSRESET

Operations

viAssertIntrSignal(vi, mode, statusID)
viAssertTrigger(vi, protocol)
viAssertUtilSignal(vi, line)
viMapTrigger(vi, trigSrc, trigDest, mode)
viUnmapTrigger(vi, trigSrc, trigDest)

SERVANT Resource

This section lists the attributes, events, and operations for the SERVANT Resource. The attributes, events, and operations in the VISA Resource Template are available to this resource in addition to the attributes and operations listed below.

Attributes

VI_ATTR_CMDR_LA
VI_ATTR_DEV_STATUS_BYTE
VI_ATTR_DMA_ALLOW_EN
VI_ATTR_FILE_APPEND_EN
VI_ATTR_GPIB_PRIMARY_ADDR
VI_ATTR_GPIB_REN_STATE
VI_ATTR_GPIB_SECONDARY_ADDR
VI_ATTR_INTF_INST_NAME
VI_ATTR_INTF_NUM
VI_ATTR_INTF_TYPE
VI_ATTR_IO_PROT
VI_ATTR_RD_BUF_OPER_MODE
VI_ATTR_SEND_END_EN
VI_ATTR_TERMCHAR
VI_ATTR_TERMCHAR_EN
VI_ATTR_TMO_VALUE
VI_ATTR_TRIG_ID
VI_ATTR_VXI_LA
VI_ATTR_VXI_VME_SYSFAIL_STATE
VI_ATTR_WR_BUF_OPER_MODE

Events

VI_EVENT_CLEAR
VI_EVENT_GPIB_LISTEN
VI_EVENT_GPIB_TALK
VI_EVENT_IO_COMPLETION
VI_EVENT_TRIG
VI_EVENT_VXI_VME_SYSRESET

Operations

viAssertIntrSignal(vi, mode, statusID)
viAssertUtilSignal(vi, line)
viBufRead(vi, buf, count, retCount)
viBufWrite(vi, buf, count, retCount)

viFlush(vi, mask)
viPrintf(vi, writeFmt, ...)
viRead(vi, buf, count, retCount)
viReadAsync(vi, buf, count, jobId)
viReadToFile(vi, fileName, count, retCount)
viScanf(vi, readFmt, ...)
viSetBuf(vi, mask, size)
viSPrintf(vi, buf, writeFmt, ...)
viSScanf(vi, buf, readFmt, ...)
viVPrintf(vi, writeFmt, params)
viVScanf(vi, readFmt, params)
viVSPrintf(vi, buf, writeFmt, params)
viVSScanf(vi, buf, readFmt, params)
viWrite(vi, buf, count, retCount)
viWriteAsync(vi, buf, count, jobId)
viWriteFromFile(vi, fileName, count, retCount)

SOCKET Resource

This section lists the attributes, events, and operations for the SOCKET Resource. The attributes, events, and operations in the VISA Resource Template are available to this resource in addition to the attributes and operations listed below.

Attributes

VI_ATTR_FILE_APPEND_EN
VI_ATTR_INTF_INST_NAME
VI_ATTR_INTF_NUM
VI_ATTR_INTF_TYPE
VI_ATTR_IO_PROT
VI_ATTR_RD_BUF_OPER_MODE
VI_ATTR_SEND_END_EN
VI_ATTR_TCPIP_ADDR
VI_ATTR_TCPIP_HOSTNAME
VI_ATTR_TCPIP_KEEPALIVE
VI_ATTR_TCPIP_NODELAY
VI_ATTR_TCPIP_PORT
VI_ATTR_TERMCHAR
VI_ATTR_TERMCHAR_EN
VI_ATTR_TMO_VALUE
VI_ATTR_TRIG_ID
VI_ATTR_WR_BUF_OPER_MODE

Events

VI_EVENT_IO_COMPLETION

Operations

viAssertTrigger(vi, protocol)
viBufRead(vi, buf, count, retCount)
viBufWrite(vi, buf, count, retCount)
viClear(vi)
viFlush(vi, mask)
viPrintf(vi, writeFmt, ...)
viQueryf(vi, writeFmt, readFmt, ...)
viRead(vi, buf, count, retCount)
viReadAsync(vi, buf, count, jobId)
viReadSTB(vi, status)
viReadToFile(vi, fileName, count, retCount)
viScanf(vi, readFmt, ...)
viSetBuf(vi, mask, size)
viSPrintf(vi, buf, writeFmt, ...)
viSScanf(vi, buf, readFmt, ...)
viVPrintf(vi, writeFmt, params)
viVQueryf(vi, writeFmt, readFmt, params)
viVScanf(vi, readFmt, params)
viVSPrintf(vi, buf, writeFmt, params)
viVSScanf(vi, buf, readFmt, params)
viWrite(vi, buf, count, retCount)
viWriteAsync(vi, buf, count, jobId)
viWriteFromFile(vi, fileName, count, retCount)

C

Technical Support Resources

This appendix describes the comprehensive resources available to you in the Technical Support section of the National Instruments Web site and provides technical support telephone numbers for you to use if you have trouble connecting to our Web site or if you do not have internet access.

NI Web Support

To provide you with immediate answers and solutions 24 hours a day, 365 days a year, National Instruments maintains extensive online technical support resources. They are available to you at no cost, are updated daily, and can be found in the Technical Support section of our Web site at `www.ni.com/support`

Online Problem-Solving and Diagnostic Resources

- **KnowledgeBase**—A searchable database containing thousands of frequently asked questions (FAQs) and their corresponding answers or solutions, including special sections devoted to our newest products. The database is updated daily in response to new customer experiences and feedback.
- **Troubleshooting Wizards**—Step-by-step guides lead you through common problems and answer questions about our entire product line. Wizards include screen shots that illustrate the steps being described and provide detailed information ranging from simple getting started instructions to advanced topics.
- **Product Manuals**—A comprehensive, searchable library of the latest editions of National Instruments hardware and software product manuals.
- **Hardware Reference Database**—A searchable database containing brief hardware descriptions, mechanical drawings, and helpful images of jumper settings and connector pinouts.
- **Application Notes**—A library with more than 100 short papers addressing specific topics such as creating and calling DLLs, developing your own instrument driver software, and porting applications between platforms and operating systems.

Software-Related Resources

- **Instrument Driver Network**—A library with hundreds of instrument drivers for control of standalone instruments via GPIB, VXI, or serial interfaces. You also can submit a request for a particular instrument driver if it does not already appear in the library.
- **Example Programs Database**—A database with numerous, non-shipping example programs for National Instruments programming environments. You can use them to complement the example programs that are already included with National Instruments products.
- **Software Library**—A library with updates and patches to application software, links to the latest versions of driver software for National Instruments hardware products, and utility routines.

Worldwide Support

National Instruments has offices located around the globe. Many branch offices maintain a Web site to provide information on local services. You can access these Web sites from `www.ni.com/worldwide`

If you have trouble connecting to our Web site, please contact your local National Instruments office or the source from which you purchased your National Instruments product(s) to obtain support.

For telephone support in the United States, dial 512 795 8248. For telephone support outside the United States, contact your local branch office:

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

Glossary

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

A

address — A string (or other language construct) that uniquely locates and identifies a resource. VISA defines an ASCII-based grammar that associates strings with particular physical devices and VISA resources.

address modifier — One of six signals in the VMEbus specifications used by VMEbus masters to indicate the address space and mode (supervisory/nonprivileged, data/program/block) in which a data transfer is to take place.

address space — In VXI/VME systems, 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* (either 16, 24, or 32) is the number of address lines required to uniquely specify a byte location in a given space. In PXI systems, the address space corresponds to 1 of 6 possible BAR locations (BAR0 through BAR5). In VME, VXI, and PXI, a given device may have addresses in one or more address spaces.

ANSI — American National Standards Institute

API — Application Programming Interface. The direct interface that an end user sees when creating an application. In VISA, the API consists of the sum of all of the operations, attributes, and events of each of the VISA Resource Classes.

ASCII — American Standard Code for Information Interchange.

asynchronous	An action or event that occurs at an unpredictable time with respect to the execution of a program.
attribute	A value within an object or resource that reflects a characteristic of its operational state.

B

b	Bit
B	Byte
backplane	In VXI/VME systems, an assembly, typically a PCB, with 96-pin connectors and signal paths that bus the connector pins. A C-size VXIbus system will have two sets of bused connectors called the J1 and J2 backplanes. A D-size VXIbus system will have three sets of bused connectors called the J1, J2, and J3 backplane.
bus error	An error that signals failed access to an address. Bus errors occur with low-level accesses to memory and usually involve hardware with bus mapping capabilities. For example, nonexistent memory, a nonexistent register, or an incorrect device access can cause a bus error.
byte order	How bytes are arranged within a word or how words are arranged within a longword. Motorola (Big-Endian) ordering stores the most significant byte (MSB) or word first, followed by the least significant byte (LSB) or word. Intel (Little-Endian) ordering stores the LSB or word first, followed by the MSB or word.

C

callback	Same as *handler*. A software routine that is invoked when an asynchronous event occurs. In VISA, callbacks can be installed on any session that processes events.
CIC	Controller-In-Charge. The device that manages the GPIB by sending interface messages to other devices.
commander	A device that has the ability to control another device. This term can also denote the unique device that has sole control over another device (as with the VXI Commander/Servant hierarchy).

communication channel	The same as *session*. A communication path between a software element and a resource. Every communication channel in VISA is unique.
configuration registers	A set of registers through which the system can identify a module device type, model, manufacturer, address space, and memory requirements. In order to support automatic system and memory configuration, the PXI and VXIbus specifications require that all PXI and VXIbus devices have a set of such registers.
controller	An entity that can control another device(s) or is in the process of performing an operation on another device.
CPU	Central processing unit

D

device	An entity that receives commands from a controller. A device can be an instrument, a computer (acting in a non-controller role), or a peripheral (such as a plotter or printer).
DLL	Dynamic Link Library. Same as a *shared library* or *shared object*. A file containing a collection of functions that can be used by multiple applications. This term is usually used for libraries on Windows platforms.
DMA	Direct memory access. High-speed data transfer between a board and memory that is not handled directly by the CPU. Not available on some systems. *See* programmed I/O.

E

embedded controller	A computer plugged directly into the VXI backplane. An example is the National Instruments VXIpc-850.
event	An asynchronous occurrence that is independent of the normal sequential execution of the process running in a system.
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

FIFO	First In-First Out; a method of data storage in which the first element stored is the first one retrieved.

G

GPIB	General Purpose Interface Bus is the common name for the communications interface system defined in ANSI/IEEE Standard 488.1-1987 and ANSI/IEEE Standard 488.2-1992.

H

handler	Same as *callback*. A software routine that is invoked when an asynchronous event occurs. In VISA, callbacks can be installed on any session that processes events.
handshaking	A type of protocol that makes it possible for two devices to synchronize operations.

I

IEEE	Institute of Electrical and Electronics Engineers
instrument	A device that accepts some form of stimulus to perform a designated task, test, or measurement function. Two common forms of stimuli are message passing and register reads and writes. Other forms include triggering or varying forms of asynchronous control.
instrument driver	A set of routines designed to control a specific instrument or family of instruments, and any necessary related files for LabWindows/CVI or LabVIEW.
interface	A generic term that applies to the connection between devices and controllers. It includes the communication media and the device/controller hardware necessary for cross-communication.
interrupt	A condition that requires attention out of the normal flow of control of a program.
I/O	input/output

L

lock	A state that prohibits sessions other than the session(s) owning the lock from accessing a resource.
logical address	An 8-bit number that uniquely identifies the location of each VXIbus device's configuration registers in a system. The A16 register address of a device is C000h + Logical Address * 40h.

M

mapping	An operation that returns a reference to a specified section of an address space and makes the specified range of addresses accessible to the requester. This function is independent of memory allocation.
message-based device	In VXI/VME systems, an intelligent device that implements the defined VXIbus registers and communication protocols. These devices are able to use Word Serial Protocol to communicate with one another through communication registers. All GPIB and Serial devices are by definition message-based, as are devices for some other interfaces. Many modern message-based devices support the IEEE 488.2 protocol.
multitasking	The ability of a computer to perform two or more functions simultaneously without interference from one another. In operating system terms, it is the ability of the operating system to execute multiple applications/processes by time-sharing the available CPU resources.

O

operation	An action defined by a resource that can be performed on a resource. In general, this term is synonymous with the connotation of the word *method* in object-oriented architectures.

P

process	An operating system element that shares a system’s resources. A multi-process system is a computer system that allows multiple programs to execute simultaneously, each in a separate process environment. A single-process system is a computer system that allows only a single program to execute at a given point in time.

programmed I/O	Low-speed data transfer between a board and memory in which the CPU moves each data value according to program instructions. *See* DMA.
protocol	Set of rules or conventions governing the exchange of information between computer systems.
PXI	PCI eXtensions for Instrumentation. PXI leverages the electrical features defined by the Peripheral Component Interconnect (PCI) specification as well as the CompactPCI form factor, which combines the PCI electrical specification with Eurocard (VME) mechanical packaging and high-performance connectors. This combination allows CompactPCI and PXI systems to have up to seven peripheral slots versus four in a desktop PCI system.

R

register	An address location that can be read from or written into or both. It may contain a value that is a function of the state of hardware or can be written into to cause hardware to perform a particular action. In other words, an address location that controls and/or monitors hardware.
register-based device	In VXI/VME systems, a servant-only device that supports only the four basic VXIbus configuration registers. Register-based devices are typically controlled by message-based devices via device-dependent register reads and writes. All PXI devices are by definition register-based, as are devices for some other interfaces.
Resource Class	The definition for how to create a particular resource. In general, this is synonymous with the connotation of the word *class* in object-oriented architectures. For VISA Instrument Control Resource Classes, this refers to the definition for how to create a resource which controls a particular capability or set of capabilities of a device.
resource or resource instance	In general, this term is synonymous with the connotation of the word *object* in object-oriented architectures. For VISA, *resource* more specifically refers to a particular implementation (or *instance* in object-oriented terms) of a Resource Class.

S

s	Second
servant	A device controlled by a Commander.
session	The same as *communication channel*. A communication path between a software element and a resource. Every communication channel in VISA is unique.
shared library or shared object	Same as *DLL*. A file containing a collection of functions that can be used by multiple applications. This term is usually used for libraries on UNIX platforms.
shared memory	A block of memory that is accessible to both a client and a server. The memory block operates as a buffer for communication. This is unique to register-based interfaces such as VXI.
SRQ	IEEE 488 Service Request. This is an asynchronous request from a remote device that requires service. A service request is essentially an interrupt from a remote device. For GPIB, this amounts to asserting the SRQ line on the GPIB. For VXI, this amounts to sending the Request for Service True event (REQT).
status byte	A byte of information returned from a remote device that shows the current state and status of the device. If the device follows IEEE 488 conventions, bit 6 of the status byte indicates whether the device is currently requesting service.
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.

T

thread	An operating system element that consists of a flow of control within a process. In some operating systems, a single process can have multiple threads, each of which can access the same data space within the process. However, each thread has its own stack and all threads can execute concurrently with one another (either on multiple processors, or by time-sharing a single processor).

V

Virtual Instrument	A name given to the grouping of software modules (in this case, VISA resources with any associated or required hardware) to give the functionality of a traditional stand-alone instrument. Within VISA, a virtual instrument is the logical grouping of any of the VISA resources.
VISA	Virtual Instrument Software Architecture. This is the general name given to this product and its associated architecture. The architecture consists of two main VISA components: the VISA Resource Manager and the VISA Instrument Control Resources.
VISA Instrument Control Resources	This is the name given to the part of VISA that defines all of the device-specific resource classes. VISA Instrument Control Resources encompass all defined device capabilities for direct, low-level instrument control.
VISA Resource Manager	This is the name given to the part of VISA that manages resources. This management includes support for finding and opening resources.
VISA Resource Template	This is the name given to the part of VISA defines the basic constraints and interface definition for the creation and use of a VISA resource. All VISA resources must derive their interface from the definition of the VISA Resource Template. This includes services for setting and retrieving attributes, receiving events, locking resources, and closing objects.
VME	Versa Module Eurocard or IEEE 1014
VXIbus	VMEbus Extensions for Instrumentation or IEEE 1155

Index

Numerics

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X