Systems Alliance

VPP-4.3.3: VISA Implementation Specification for the G Language

February 26, 2016

Revision 5.7

Systems Alliance

VPP-4.3.3 Revision History

This section is an overview of the revision history of the VPP-4.3.3 specification.

Revision 1.0, December 29, 1995

Original VISA document. Changes from VISA Transition Library include bindings for locking, asynchronous I/O, 32-bit register access, block moves, shared memory operations, and serial interface support.

Revision 1.1, January 22, 1997

Added new attributes, error codes, events, and formatted I/O buffers.

Revision 2.0, January 9, 1998

Added error handling event, more formatted I/O operations, more serial attributes and extended searching capabilities.

Revision 2.0.1, December 4, 1998

Added new modes to give more robust functionality to viGpibControlREN. Updated information regarding contacting the Alliance.

Revision 2.2, November 19, 1999

Added new resource classes for GPIB (INTFC and SERVANT), VXI (BACKPLANE and SERVANT), and TCPIP (INSTR, SOCKET, and SERVANT).

Revision 3.0 Draft, January 14, 2003

Added new resource class for USB (INSTR).

Revision 3.0, January 15, 2004

Approved at IVI Board of Directors meeting.

Revision 4.0 Draft, October 4, 2005

Added new resource class for PXI (INSTR) to incorporate PXISA extensions. Added 64-bit extensions for register-based operations. Added support for Win64.

Revision 4.1, February 14, 2008

Updated the introduction to reflect the IVI Foundation organization changes. Replaced Notice with text used by IVI Foundation specifications.

Revision 4.1, April 14, 2008

Editorial change to update the IVI Foundation contact information in the Important Information section to remove obsolete address information and refer only to the IVI Foundation web site.

Revision 5.0, June 9, 2010

Added several new TCPIP INSTR attributes regarding HiSLIP devices.

Revision 5.7, February 26, 2016

Added PXI trigger lines TTL8-TTL11. Added existing VXI trigger lines.

NOTICE

VPP-4.3.3: *VISA Implementation Specification for the G Language* is authored by the IVI Foundation member companies. For a vendor membership roster list, please visit the IVI Foundation web site at `www.ivifoundation.org`.

The IVI Foundation wants to receive your comments on this specification. You can contact the Foundation through the web site at `www.ivifoundation.org`.

Warranty

The IVI Foundation and its member companies make no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The IVI Foundation and its member companies shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Trademarks

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

No investigation has been made of common-law trademark rights in any work.

Table of Contents

Figure

Tables

Section 1 Introduction to the VXI*plug&play* Systems Alliance and the IVI Foundation

The VXI*plug&play* Systems Alliance was founded by members who shared a common commitment to end-user success with open, multivendor VXI systems. The alliance accomplished major improvements in ease of use by endorsing and implementing common standards and practices in both hardware and software, beyond the scope of the VXIbus specifications. The alliance used both formal and de facto standards to define complete system frameworks. These standard frameworks gave end-users "plug & play" interoperability at both the hardware and system software level.

The IVI Foundation is an organization whose members share a common commitment to test system developer success through open, powerful, instrument control technology. The IVI Foundation's primary purpose is to develop and promote specifications for programming test instruments that simplify interchangeability, provide better performance, and reduce the cost of program development and maintenance.

In 2002, the VXI*plug&play* Systems Alliance voted to become part of the IVI Foundation. In 2003, the VXI*plug&play* Systems Alliance formally merged into the IVI Foundation. The IVI Foundation has assumed control of the VXI*plug&play* specifications, and all ongoing work will be accomplished as part of the IVI Foundation.

All references to VXI*plug&play* Systems Alliance within this document, except contact information, were maintained to preserve the context of the original document.

Section 2 Overview of VISA Implementation Specification

This section introduces the VISA Implementation Specification for the G Language. This specification is a document authored by the VXI*plug&play* Systems Alliance. The technical work embodied in this document and the writing of this document was performed by the VISA Technical Working Group.

This section provides a complete overview of the VISA implementation specification, and gives readers general information that may be required to understand how to read, interpret, and implement individual aspects of this specification. This section is organized as follows:

- Objectives of the specification
- Scope and organization of this specification
- Application of this specification
- References
- Definitions of terms and acronyms
- Conventions
- Communication

2.1 Objectives of This Specification

VISA gives VXI and GPIB software developers, particularly instrument driver developers, the functionality needed by instrument drivers in an interface-independent fashion for MXI, embedded VXI, GPIB-VXI, GPIB, and asynchronous serial controllers. VXI*plug&play* drivers written to the VISA specifications can execute on VXI*plug&play* system frameworks that have the VISA I/O library.

The VISA specification provides a common standard for the VXI*plug&play* System Alliance for developing multi-vendor software programs, including instrument drivers. This specification describes the VISA software model and the VISA Application Programming Interface (API).

The VISA Implementation Specification for the G Language addresses particular issues related to implementing source and binary level compatibility within G Language framework systems. Implementation issues for textual languages are described in VPP-4.3.2: *VISA Implementation Specification for Textual Languages*.

2.2 Audience for This Specification

There are three audiences for this specification. The first audience is instrument driver developers--whether an instrument vendor, system integrator, or end user--who want to implement instrument driver software that is compliant with the VXI*plug&play* standards. The second audience is I/O vendors who want to implement VISA-compliant I/O software. The third audience is instrumentation end users and application programmers who want to implement applications that utilize instrument drivers compliant with this specification.

2.3 Scope and Organization of This Specification

This specification is organized in sections, with each section discussing a particular aspect of the VISA model.

Section 1 explains the VXI*plug&play* Systems Alliance and its relation to the IVI Foundation.

Section 2 provides an overview of this specification, including the objectives, scope and organization, application, references, definition of terms and acronyms, and conventions.

Section 3 provides the details of the VISA bindings to G Language framework systems.

2.4 Application of This Specification

This specification is intended for use by developers of VXI*plug&play* instrument drivers and by developers of VISA I/O software. It is also useful as a reference for end users of VXI*plug&play* instrument drivers. This specification is intended to be used in conjunction with the VPP-3.x specifications, including the *Instrument Drivers Architecture and Design Specification* (VPP-3.1), the *Instrument Driver Functional Body Specification* (VPP-3.2), the *Instrument Interactive Developer Interface Specification* (VPP-3.3), and the *Instrument Driver Programmatic Developer Interface Specification* (VPP-3.4). These related specifications describe the implementation details for specific instrument drivers that are used with specific system frameworks. VXI*plug&play* instrument drivers developed in accordance with these specifications can be used in a wide variety of higher-level software environments, as described in the *System Frameworks Specification* (VPP-2).

2.5 References

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

- 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 1174-2000, Standard Serial Interface for Programmable Instrumentation*
- VPP-1, VXI*plug&play* Charter Document
- VPP-2, *System Frameworks Specification*
- VPP-3.1, *Instrument Drivers Architecture and Design Specification*
- VPP-3.2, *Instrument Driver Functional Body Specification*
- VPP-3.3, *Instrument Driver Interactive Developer Interface Specification*
- VPP-3.4, *Instrument Driver Programmatic Developer Interface Specification*
- VPP-4.3, *The VISA Library*
- VPP-4.3.2, *VISA Implementation Specification for Textual Languages*
- VPP-6, *Installation and Packaging Specification*

- VPP-7, *Soft Front Panel Specification*
- VPP-9, *Instrument Vendor Abbreviations*
- VXI-1, *VXIbus System Specification*, Revision 1.4, VXIbus Consortium

2.6 Definition of Terms and Acronyms

The following are some commonly used terms within this document.

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 or interfaces and VISA resources.
ADE	Application Development Environment
API	Application Programmers 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.
Attribute	A value within a resource that reflects a characteristic of the operational state of a resource.
Communication Channel	The same as *Session*. A communication path between a software element and a resource. Every communication channel in VISA is unique.
Controller	A device that can control another device(s) or is in the process of performing an operation on another device.
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). In VISA, the concept of a device is generally the logical association of several VISA resources.
G	Graphical language used to describe programs in LabVIEW.
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.
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.
Instrument Driver	Library of functions for controlling a specific instrument
LabVIEW	Graphical programming ADE
LLB	LabVIEW VI library
Operation	An action defined by a resource that can be performed on a resource.

Process	An operating system component 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.
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 that controls a particular capability 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. In VISA, every defined software module is a resource.
Session	The same as *Communication Channel*. A communication path between a software element and a resource. Every communication channel in VISA is unique.
SRQ	IEEE 488 Service Request. This is an asynchronous request from a remote GPIB 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 if the device is currently requesting service.
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. The VISA Instrument Control Resources Organizer serves as a means to group any number of any type of VISA Instrument Control Resources within a VISA system.
VI	LabVIEW program or Virtual Instrument
VISA	Virtual Instrument Software Architecture. This is the general name given to this document 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 and interface capabilities for direct, low-level instrument control.

2.7 Conventions

Throughout this specification you will see the following headings on certain paragraphs. These headings instill special meaning on these paragraphs.

Rules must be followed to ensure compatibility with the System Framework. A rule is characterized by the use of the words **SHALL** and **SHALL NOT** in bold upper case characters. These words are not used in this manner for any other purpose other than stating rules.

Observations spell out implications of rules and bring attention to things that might otherwise be overlooked. They also give the rationale behind certain rules, so that the reader understands why the rule must be followed.

A Note on the text of the specification: Any text which appears without heading should be considered as description of the standard and how the architecture was intended to operate. The purpose of this text is to give the reader a deeper understanding of the intentions of the specification including the underlying model and specific required features. As such, the implementor of this standard should take great care to ensure that a particular implementation does not conflict with the text of the standard.

Section 3 VISA Framework Bindings

3.1 Type Assignments

Table 3.1.1 gives the type assignments for LabVIEW for each type defined in VPP-4.3.

Table 3.1.1 Type Assignments

VISA Data Type	LabVIEW		Description
`ViUInt64`		input	A 64-bit unsigned integer.
`ViPUInt64`		output	The location of a 64-bit unsigned integer.
`ViInt64`		input	A 64-bit signed integer.
`ViPInt64`		output	The location of a 64-bit signed integer.
`ViUInt32`		input	A 32-bit unsigned integer.
`ViPUInt32`		output	The location of a 32-bit unsigned integer.
`ViInt32`		input	A 32-bit signed integer.
`ViPInt32`		output	The location of a 32-bit signed integer.
`ViUInt16`		input	A 16-bit unsigned integer.
`ViPUInt16`		output	The location of a 16-bit unsigned integer.
`ViInt16`		input	A 16-bit signed integer.
`ViPInt16`		output	The location of a 16-bit signed integer.
`ViUInt8`		input	An 8-bit unsigned integer.
`ViPUInt8`		output	The location of an 8-bit unsigned integer.
`ViInt8`		input	An 8-bit signed integer.
`ViPInt8`		output	The location of an 8-bit signed integer.
`ViAddr`		input	A type that references another data type, in cases where the other data type may vary depending on a particular context.
`ViPAddr`		output	The location of a `ViAddr`.
`ViChar`		input	An 8-bit integer representing an ASCII character.
`ViPChar`		output	The location of a `ViChar`.
`ViByte`		input	An 8-bit unsigned integer representing an extended ASCII character.

(continues)

Table 3.1.1 Type Assignments (Continued)

VISA Data Type	LabVIEW	Description
`ViPByte`	 output	The location of a `ViByte`.
`ViBoolean`	 input	A type for which there are exactly two complementary values: `VI_TRUE` and `VI_FALSE`.
`ViPBoolean`	 output	The location of a `ViBoolean`.
`ViReal32`	 input	A 32-bit single-precision value.
`ViPReal32`	 output	The location of a 32-bit single-precision value.
`ViReal64`	 input	A 64-bit double-precision value.
`ViPReal64`	 output	The location of a 64-bit double-precision value.
`ViBuf`	 input	The location of a block of data.
`ViPBuf`	 output	The location to store a block of data.
`ViString`	 input	The location of a NULL-terminated ASCII string.
`ViPString`	 output	The location to store a NULL-terminated ASCII string.
`ViRsrc`	 input	A type that is further restricted to adhere to the addressing grammar for resources as presented in Section 3 of VPP-4.3.
`ViPRsrc`	 output	The location of a `ViRsrc`.
`ViStatus`	 input	Error cluster containing:  Error indicator.  VISA-defined Completion and Error termination codes.  Name of VI in which the error occurred.
`ViPStatus`	 output	Error cluster containing:  Error indicator.  VISA-defined Completion and Error termination codes.  Name of VI in which the error occurred.

(continues)

Table 3.1.1 Type Assignments (Continued)

VISA Data Type	LabVIEW		Description
`ViVersion`		input	A defined type that contains a reference to all information necessary for the architect to represent the current version of a resource.
`ViPVersion`		output	The location of a `ViVersion`.
`ViObject`		input	The most fundamental VISA data type. It contains attributes and can be closed when no longer needed.
`ViPObject`		output	The location of a `ViObject`.
`ViSession`		input	A defined type that contains a reference to all information necessary for the architect to manage a communication channel with a resource.
`ViPSession`		output	The location of a `ViSession`.
`ViAccessMode`		input	A defined type that specifies the different mechanisms that control access to a resource.
`ViBusAddress`		input	A type that represents the system dependent physical address.
`ViPBusAddress`		output	The location of a `ViBusAddress`.
`ViBusSize`		input	A type that represents the system dependent physical address size.
`ViAttrState`	**Note 1**	input	A value unique to the individual type of an attribute.
`ViPAttrState`	**Note 1**	output	The location of a `ViAttrState`.
`ViEventType`		input	A defined type that uniquely identifies the type of an event.
`ViPEventType`		output	The location of a `ViEventType`.
`ViEvent`		input	A defined type that encapsulates the information necessary to process an event.
`ViPEvent`		output	The location of a `ViEvent`.
`ViKeyId`		input	A defined type that contains a reference to all information necessary for the architect to manage the association of a thread or process and session with a lock on a resource.
`ViPKeyId`		output	The location of a `ViKeyId`.
`ViConstString`		input	A `ViString` type that is guaranteed to not be modified by any driver.

Note 1: ***The size of this VISA data type is dependent on the attribute.***

RULE 3.1.1

All types in Table 3.1.1 **SHALL** be defined to the specified bindings.

3.2 Operation Prototypes

The following section specifies the operation prototypes for LabVIEW.

3.2.1 Common Controls and Indicators

Each VISA VI has an Error In and an Error Out terminal defined on its connector pane in the lower left and lower right terminals, respectively. The error clusters are used to report all VISA completion and error codes. Inside the cluster, a Boolean error indicator, a numeric error code, and an error source string indicator report if there is an error, identify the specific error condition, and list the source (name) of the VI in which the error occurred.

Unless otherwise noted, a VISA function will not attempt to operate if an error condition is passed in through the Error In cluster. In this case, the function passes the contents of Error In cluster out the Error Out cluster. If no error condition is passed in through the Error In cluster, the Error Out cluster represents the status of that VI. By wiring the Error Out cluster of one VI to the Error In cluster of another VI, users can pass error information throughout their program that will propagate to the top-level VI in their application. A benefit of error input/output is that data dependency is added to VIs that are not otherwise data dependent, thus adding a means of specifying execution order beyond traditional sequence structures.

With the exception of `viFindRsrc`, `viOpen`, and `viClose`, all VISA VIs have a VISA refnum control and the dup VISA refnum indicator. `viOpen` has only a VISA refnum indicator and `viClose` has only a VISA refnum control. The VISA refnum control and the dup VISA refnum indicator are defined in the connector pane in the upper left and upper right terminals, respectively. Just like the error input/output clusters, the VISA refnum control and the dup VISA refnum indicator can be used to specify data dependency of a program using the VISA resource. For VIs that have both the VISA refnum control and the dup VISA refnum indicator, the value of the indicator is equal to the value passed into the control.

3.2.2 Scope of Functionality

RULE 3.2.1

IF a VISA implementation complies with the GWINNT framework, **THEN** it **SHALL** perform instrument I/O by accessing the `visa32.dll` from the WINNT framework.

OBSERVATION 3.2.1

While the VISA library for each G-based framework uses the library provided by the corresponding text-based language framework, the following differences in functionality exist:

- The formatted I/O functions are not exported.
- Receiving events by callback is not exported. (Queuing events is exported.)
- The operations `viOpenDefaultRM, viFindNext, viParseRsrc,` and `viParseRsrcEx` are not exported.
- `viOpen` includes the functionality of `viOpenDefaultRM` and `viParseRsrc.`
- `viFindRsrc` combines the functionality of `viOpenDefaultRM`, `viFindRsrc`, `viFindNext`, and `viClose` into a single operation.

RULE 3.2.2

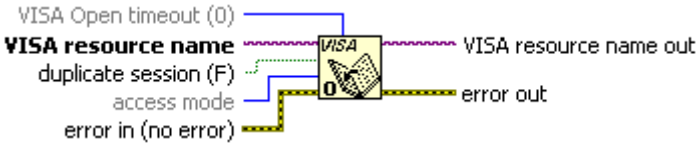
All functions and operations specified in Section 3.2 **SHALL** be implemented as specified.

3.2.3 viFindRsrc (VISA Find Resource)

Inputs	
abc	expression
	error in

Outputs	
abc	find list
U32	return count
	error out

3.2.4 viOpen (VISA Open)

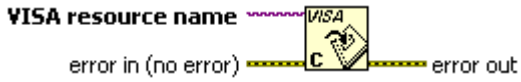


Inputs	
U32	VISA Open timeout
I/O	VISA resource name
TF	duplicate session (F)
U32	access mode
	error in

Outputs	
I/O	VISA resource name out
	error out

Note: ***If "duplicate session" is False and a session to the resource is already open, then*** `viOpen` ***is not called.***

3.2.5 viClose (VISA Close)

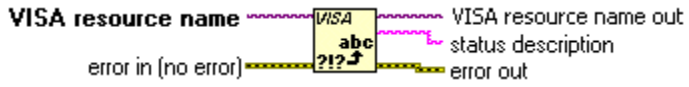


Inputs	
I/O	VISA resource name
	error in

Outputs	
	error out

Note: `viClose` ***will execute even if an error condition is passed in through the error in cluster.***

3.2.6 viStatusDesc (VISA Status Description)

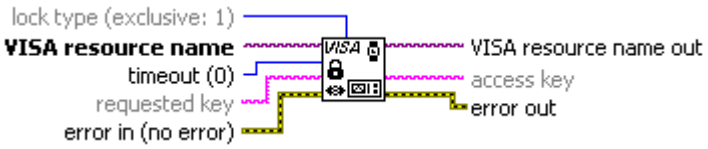


Inputs	
I/O	VISA resource name
	error in

Outputs	
I/O	VISA resource name out
abc	status description
	error out

Note: `viStatusDesc` ***will execute even if an error condition is passed in through the error in cluster.***

3.2.7 viLock (VISA Lock Async)



Inputs	
U32	lock type
I/O	VISA resource name
U32	timeout
abc	requested key
	error in

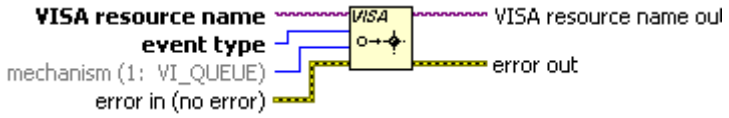
Outputs	
I/O	VISA resource name out
abc	access key
	error out

3.2.8 viUnlock (VISA Unlock)

Inputs	
I/O	VISA resource name
	error in

Outputs	
I/O	VISA resource name out
	error out

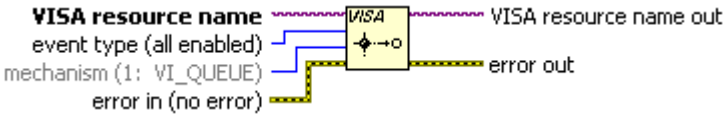
3.2.9 viEnableEvent (VISA Enable Event)



Inputs	
I/O	VISA resource name
U32	event type
U16	mechanism
	error in

Outputs	
I/O	VISA resource name out
	error out

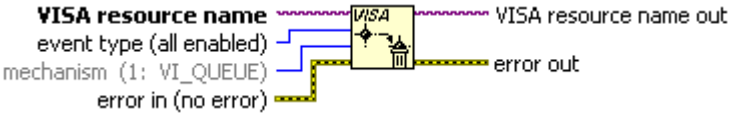
3.2.10 viDisableEvent (VISA Disable Event)



Inputs	
I/O	VISA resource name
U32	event type
U16	mechanism
	error in

Outputs	
I/O	VISA resource name out
	error out

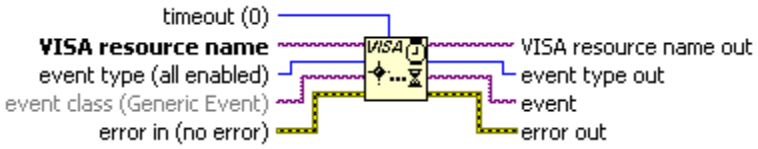
3.2.11 viDiscardEvents (VISA Discard Events)



Inputs	
I/O	VISA resource name
U32	event type
U16	mechanism
	error in

Outputs	
I/O	VISA resource name out
	error out

3.2.12 viWaitOnEvent (VISA Wait on Event)



Inputs	
U32	timeout
I/O	VISA resource name
U32	event type
I/O	event class
	error in

Outputs	
I/O	VISA resource name out
U32	event type out
I/O	event
	error out

3.2.13 viRead (VISA Read)

Inputs	
I/O	VISA resource name
U32	byte count
	error in

Outputs	
I/O	VISA resource name out
abc	read buffer
U32	return count
	error out

3.2.14 viWrite (VISA Write)

Inputs	
I/O	VISA resource name
abc	write buffer
	error in

Outputs	
I/O	VISA resource name out
U32	return count
	error out

3.2.15 viAssertTrigger (VISA Assert Trigger)

Inputs	
I/O	VISA resource name
U16	protocol
	error in

Outputs	
I/O	VISA resource name out
	error out

3.2.16 viReadSTB (VISA Read STB)

Inputs	
I/O	VISA resource name
	error in

Outputs	
I/O	VISA resource name out
U16	status byte
	error out

3.2.17 viClear (VISA Clear)

Inputs	
I/O	VISA resource name
	error in

Outputs	
I/O	VISA resource name out
	error out

3.2.18 viSetBuf (VISA Set I/O Buffer Size)

Inputs	
I/O	VISA resource name
U16	mask
U32	size
	error in

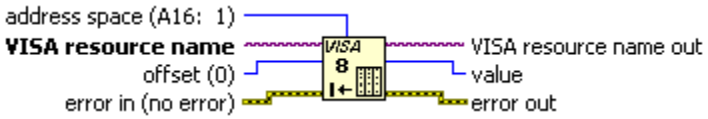
Outputs	
I/O	VISA resource name out
	error out

3.2.19 viFlush (VISA Flush I/O Buffer)

Inputs	
I/O	VISA resource name
U16	mask
	error in

Outputs	
I/O	VISA resource name out
	error out

3.2.20 viIn8/viIn16/viIn32/viIn64/ viIn8Ex/viIn16Ex/viIn32Ex/viIn64Ex (VISA In 8/VISA In 16/VISA In 32/VISA In 64)



viIn16

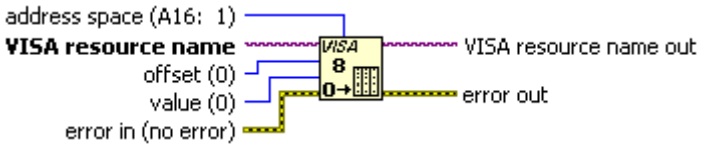
viIn32

viIn64

Inputs	
U16	address space
I/O	VISA resource name
U32	offset (Use the 64-bit control for the Ex operations.)
	error in

Outputs	
I/O	VISA resource name out
U8	value (Use the 16-bit indicator for `viIn16`, the 32-bit indicator for `viIn32`, or the 64-bit indicator for `viIn64`.)
	error out

3.2.21 viOut8/viOut16/viOut32/viOut64/ viOut8Ex/viOut16Ex/viOut32Ex/viOut64Ex (VISA Out 8/VISA Out 16/VISA Out 32/VISA Out 64)



viOut16

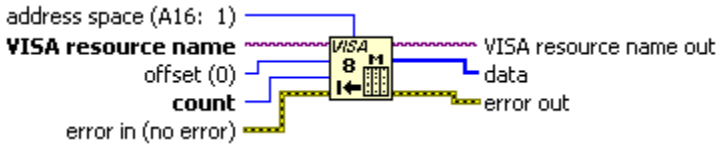
viOut32

viOut64

Inputs	
U16	address space
I/O	VISA resource name
U32	offset (Use the 64-bit control for the Ex operations.)
U8	value (Use the 16-bit control for `viOut16`, the 32-bit control for `viOut32`, or the 64-bit control for `viOut64`.)
	error in

Outputs	
I/O	VISA resource name out
	error out

3.2.22 viMoveIn8/viMoveIn16/viMoveIn32/viMoveIn64/ viMoveIn8Ex/viMoveIn16Ex/viMoveIn32Ex/viMoveIn64Ex (VISA MoveIn 8/VISA MoveIn 16/VISA MoveIn 32/VISA MoveIn 64)



viMoveIn16

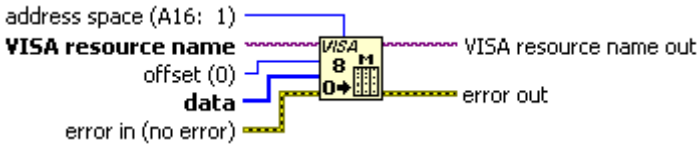
viMoveIn32

viMoveIn64

Inputs	
U16	address space
I/O	VISA resource name
U32	offset (Use the 64-bit control for the Ex operations.)
U32	count
	error in

Outputs	
I/O	VISA resource name out
U8	data (Use the 16-bit indicator for `viMoveIn16`, the 32-bit indicator for `viMoveIn32`, or the 64-bit indicator for `viMoveIn64`.)
	error out

3.2.23 viMoveOut8/viMoveOut16/viMoveOut32/viMoveOut64/ viMoveOut8Ex/viMoveOut16Ex/viMoveOut32Ex/viMoveOut64Ex (VISA MoveOut 8/VISA MoveOut 16/VISA MoveOut 32/VISA MoveOut 64)

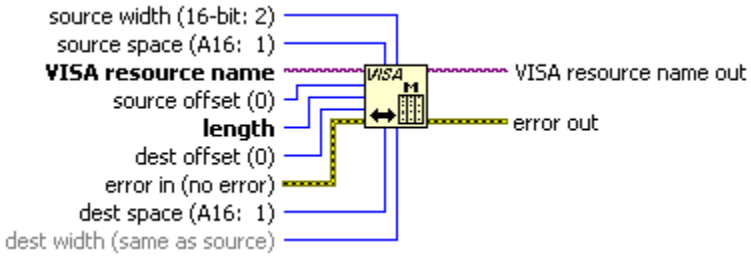


viMoveOut16 **viMoveOut32** **viMoveOut64**

Inputs	
U16	address space
I/O	VISA resource name
U32	offset (Use the 64-bit control for the Ex operations.)
[U8]	data (Use the 16-bit control for `viMoveOut16`, the 32-bit control for `viMoveOut32`, or the 64-bit control for `viMoveOut64`.)
	error in

Outputs	
I/O	VISA resource name out
	error out

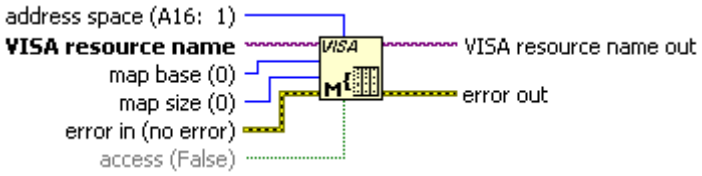
3.2.24 viMove/viMoveEx (VISA Move)



Inputs	
U16	source width
U16	source space
I/O	VISA resource name
U32	source offset
U32	length
U32	dest offset (Use the 64-bit control for the Ex operations.)
	error in
U16	dest space
U16	dest width

Outputs	
I/O	VISA resource name out
	error out

3.2.25 viMapAddress/viMapAddressEx (VISA Map Address)



Inputs	
U16	address space
I/O	VISA resource name
U32	map base
U32	map size
	error in
TF	access

Outputs	
I/O	VISA resource name out
	error out

3.2.26 viUnmapAddress (VISA Unmap Address)

Inputs	
I/O	VISA resource name
	error in

Outputs	
I/O	VISA resource name out
	error out

3.2.27 viPeek8/viPeek16/viPeek32/viPeek64 (VISA Peek 8/VISA Peek 16/VISA Peek 32/VISA Peek 64)

viPeek16

viPeek32

viPeek64

Inputs	
I/O	VISA resource name
U32	offset
	error in

Outputs	
I/O	VISA resource name out
U8	value (Use the 16-bit indicator for `viPeek16`, the 32-bit indicator for `viPeek32`, or the 64-bit indicator for `viPeek64`.)
	error out

3.2.28 viPoke8/viPoke16/viPoke32/viPoke64 (VISA Poke 8/VISA Poke 16/VISA Poke 32/VISA Poke 64)

viPoke16

viPoke32

viPoke64

Inputs	
I/O	VISA resource name
U32	offset
U8	value (Use the 16-bit control for `viPoke16`, the 32-bit control for `viPoke32`, or the 64-bit control for viPoke64.)
	error in

Outputs	
I/O	VISA resource name
	error out

3.2.29 viMemAlloc (VISA Memory Allocation)

Inputs	
I/O	VISA resource name
U32	size
	error in

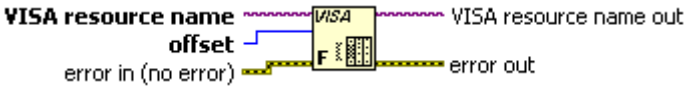
Outputs	
I/O	VISA resource name out
U32	offset
	error out

3.2.30 viMemAllocEx (VISA Memory Allocation Ex)

Inputs	
I/O	VISA resource name
U32	size
	error in

Outputs	
I/O	VISA resource name out
U64	offset
	error out

3.2.31 viMemFree/viMemFreeEx (VISA Memory Free)



Inputs	
I/O	VISA resource name
U32	offset (Use the 64-bit control for the Ex operations.)
	error in

Outputs	
I/O	VISA resource name out
	error out

3.2.32 viReadToFile (VISA Read To File)

Inputs	
I/O	VISA resource name
path	filename
U32	byte count
error	error in

Outputs	
I/O	VISA resource name out
U32	return count
error	error out

3.2.33 viWriteFromFile (VISA Write From File)

Inputs	
I/O	VISA resource name
path	filename
U32	count
error	error in

Outputs	
I/O	VISA resource name out
U32	return count
error	error out

3.2.34 viGpibControlREN (VISA GPIB Control REN)

Inputs	
I/O	VISA resource name
U16	mode
error	error in

Outputs	
I/O	VISA resource name out
error	error out

3.2.35 viGpibControlATN (VISA GPIB Control ATN)

Inputs	
I/O	VISA resource name
U16	mode
	error in

Outputs	
I/O	VISA resource name out
	error out

3.2.36 viGpibSendIFC (VISA GPIB Send IFC)

Inputs	
I/O	VISA resource name
	error in

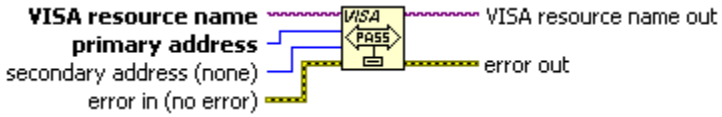
Outputs	
I/O	VISA resource name out
	error out

3.2.37 viGpibCommand (VISA GPIB Command)

Inputs	
I/O	VISA resource name
abc	command
	error in

Outputs	
I/O	VISA resource name out
U32	return count
	error out

3.2.38 viGpibPassControl (VISA GPIB Pass Control)



Inputs	
I/O	VISA resource name
U16	primary address
U16	secondary address
	error in

Outputs	
I/O	VISA resource name out
	error out

3.2.39 viVxiCommandQuery (VISA VXI Cmd or Query)

Inputs	
I/O	VISA resource name
U16	mode
U32	command
	error in

Outputs	
I/O	VISA resource name out
U32	response
	error out

3.2.40 viAssertUtilSignal (VISA Assert Utility Signal)

Inputs	
I/O	VISA resource name
U16	bus signal
	error in

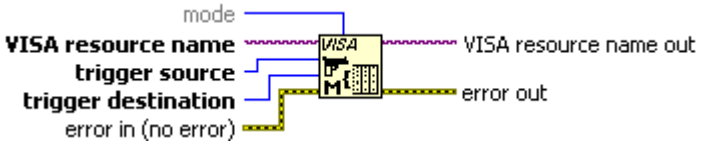
Outputs	
I/O	VISA resource name out
	error out

3.2.41 viAssertIntrSignal (VISA Assert Interrupt Signal)

Inputs	
I/O	VISA resource name
I16	mode
U32	status ID
	error in

Outputs	
I/O	VISA resource name out
	error out

3.2.42 viMapTrigger (VISA Map Trigger)



Inputs	
U16	mode
I/O	VISA resource name
I16	trigger source
I16	trigger destination
	error in

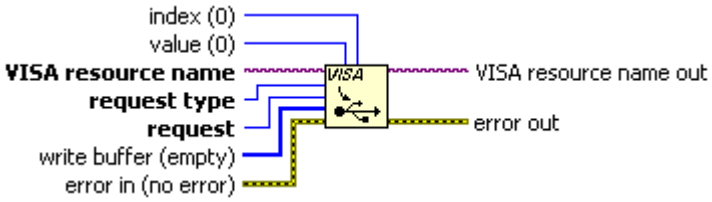
Outputs	
I/O	VISA resource name out
	error out

3.2.43 viUnmapTrigger (VISA Unmap Trigger)

Inputs	
I/O	VISA resource name
I16	trigger source
U16	trigger destination
	error in

Outputs	
I/O	VISA resource name out
	error out

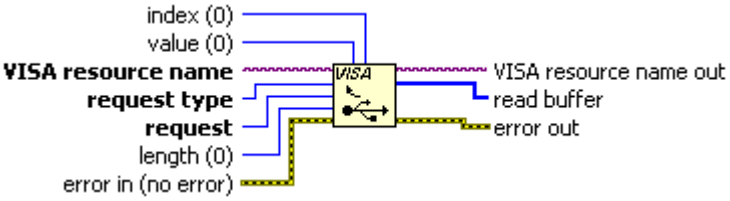
3.2.44 viUsbControlOut (VISA USB Control Out)



Inputs	
U16	index
U16	value
I/O	VISA resource name
I16	request type
I16	request
[U8]	write buffer
	error in

Outputs	
I/O	VISA resource name out
	error out

3.2.45 viUsbControlIn (VISA USB Control In)



Inputs	
U16	index
U16	value
I/O	VISA resource name
I16	request type
I16	request
U16	length
	error in

Outputs	
I/O	VISA resource name out
[U8]	read buffer
	error out

3.3 Completion and Error Codes

Table 3.3.1 lists the Completion and Error codes defined for all G-based framework bindings.

Table 3.3.1. Completion and Error Codes

Completion and Error Codes	Values
VI_SUCCESS	0
VI_SUCCESS_EVENT_EN	3FFF0002h
VI_SUCCESS_EVENT_DIS	3FFF0003h
VI_SUCCESS_QUEUE_EMPTY	3FFF0004h
VI_SUCCESS_TERM_CHAR	3FFF0005h
VI_SUCCESS_MAX_CNT	3FFF0006h
VI_SUCCESS_DEV_NPRESENT	3FFF007Dh
VI_SUCCESS_TRIG_MAPPED	3FFF007Eh
VI_SUCCESS_QUEUE_NEMPTY	3FFF0080h
VI_SUCCESS_NCHAIN	3FFF0098h
VI_SUCCESS_NESTED_SHARED	3FFF0099h
VI_SUCCESS_NESTED_EXCLUSIVE	3FFF009Ah
VI_SUCCESS_SYNC	3FFF009Bh
VI_WARN_QUEUE_OVERFLOW	3FFF000Ch
VI_WARN_CONFIG_NLOADED	3FFF0077h
VI_WARN_NULL_OBJECT	3FFF0082h
VI_WARN_NSUP_ATTR_STATE	3FFF0084h
VI_WARN_UNKNOWN_STATUS	3FFF0085h
VI_WARN_NSUP_BUF	3FFF0088h
VI_WARN_EXT_FUNC_NIMPL	3FFF00A9h
VI_ERROR_SYSTEM_ERROR	BFFF0000h
VI_ERROR_INV_OBJECT	BFFF000Eh
VI_ERROR_INV_SESSION	BFFF000Eh
VI_ERROR_RSRC_LOCKED	BFFF000Fh
VI_ERROR_INV_EXPR	BFFF0010h
VI_ERROR_RSRC_NFOUND	BFFF0011h
VI_ERROR_INV_RSRC_NAME	BFFF0012h
VI_ERROR_INV_ACC_MODE	BFFF0013h
VI_ERROR_TMO	BFFF0015h
VI_ERROR_CLOSING_FAILED	BFFF0016h
VI_ERROR_INV_DEGREE	BFFF001Bh
VI_ERROR_INV_JOB_ID	BFFF001Ch

(continues)

Table 3.3.1. Completion and Error Codes (Continued)

Completion and Error Codes	Values
VI_ERROR_NSUP_ATTR_STATE	BFFF001Eh
VI_ERROR_ATTR_READONLY	BFFF001Fh
VI_ERROR_INV_LOCK_TYPE	BFFF0020h
VI_ERROR_INV_ACCESS_KEY	BFFF0021h
VI_ERROR_INV_EVENT	BFFF0026h
VI_ERROR_INV_MECH	BFFF0027h
VI_ERROR_HNDLR_NINSTALLED	BFFF0028h
VI_ERROR_INV_HNDLR_REF	BFFF0029h
VI_ERROR_INV_CONTEXT	BFFF002Ah
VI_ERROR_NENABLED	BFFF002Fh
VI_ERROR_ABORT	BFFF0030h
VI_ERROR_RAW_WR_PROT_VIOL	BFFF0034h
VI_ERROR_RAW_RD_PROT_VIOL	BFFF0035h
VI_ERROR_OUTP_PROT_VIOL	BFFF0036h
VI_ERROR_INP_PROT_VIOL	BFFF0037h
VI_ERROR_BERR	BFFF0038h
VI_ERROR_IN_PROGRESS	BFFF0039h
VI_ERROR_INV_SETUP	BFFF003Ah
VI_ERROR_QUEUE_ERROR	BFFF003Bh
VI_ERROR_ALLOC	BFFF003Ch
VI_ERROR_INV_MASK	BFFF003Dh
VI_ERROR_IO	BFFF003Eh
VI_ERROR_INV_FMT	BFFF003Fh
VI_ERROR_NSUP_FMT	BFFF0041h
VI_ERROR_LINE_IN_USE	BFFF0042h
VI_ERROR_NSUP_MODE	BFFF0046h
VI_ERROR_SRQ_NOCCURRED	BFFF004Ah
VI_ERROR_INV_SPACE	BFFF004Eh
VI_ERROR_INV_OFFSET	BFFF0051h
VI_ERROR_INV_WIDTH	BFFF0052h
VI_ERROR_NSUP_OFFSET	BFFF0054h
VI_ERROR_NSUP_VAR_WIDTH	BFFF0055h
VI_ERROR_WINDOW_NMAPPED	BFFF0057h
VI_ERROR_RESP_PENDING	BFFF0059h
VI_ERROR_NLISTENERS	BFFF005Fh

(continues)

Table 3.3.1. Completion and Error Codes (Continued)

Completion and Error Codes	Values
VI_ERROR_NSUP_ATTR	BFFF001Dh
VI_ERROR_NCIC	BFFF0060h
VI_ERROR_NSYS_CNTLR	BFFF0061h
VI_ERROR_NSUP_OPER	BFFF0067h
VI_ERROR_INTR_PENDING	BFFF0068h
VI_ERROR_ASRL_PARITY	BFFF006Ah
VI_ERROR_ASRL_FRAMING	BFFF006Bh
VI_ERROR_ASRL_OVERRUN	BFFF006Ch
VI_ERROR_TRIG_NMAPPED	BFFF006Eh
VI_ERROR_NSUP_ALIGN_OFFSET	BFFF0070h
VI_ERROR_USER_BUF	BFFF0071h
VI_ERROR_RSRC_BUSY	BFFF0072h
VI_ERROR_NSUP_WIDTH	BFFF0076h
VI_ERROR_INV_PARAMETER	BFFF0078h
VI_ERROR_INV_PROT	BFFF0079h
VI_ERROR_INV_SIZE	BFFF007Bh
VI_ERROR_WINDOW_MAPPED	BFFF0080h
VI_ERROR_NIMPL_OPER	BFFF0081h
VI_ERROR_INV_LENGTH	BFFF0083h
VI_ERROR_INV_MODE	BFFF0091h
VI_ERROR_SESN_NLOCKED	BFFF009Ch
VI_ERROR_MEM_NSHARED	BFFF009Dh
VI_ERROR_LIBRARY_NFOUND	BFFF009Eh
VI_ERROR_NSUP_INTR	BFFF009Fh
VI_ERROR_INV_LINE	BFFF00A0h
VI_ERROR_FILE_ACCESS	BFFF00A1h
VI_ERROR_FILE_IO	BFFF00A2h
VI_ERROR_NSUP_LINE	BFFF00A3h
VI_ERROR_NSUP_MECH	BFFF00A4h
VI_ERROR_INTF_NUM_NCONFIG	BFFF00A5h
VI_ERROR_CONN_LOST	BFFF00A6h

OBSERVATION 3.3.1

Notice that all success and warning codes (Completion codes) have a value that is greater than or equal to 0, while all Error codes have a value that is less than 0. Therefore, an application should determine whether an invocation of a given operation fails by checking to see whether the return value is *less than* 0 (as opposed to *not equal to* 0). Applications using the G language can also just check the *status* member of the error cluster output, which is set whenever an error has occurred.

3.4 Attribute Operations

The Property Node, shown in Figure 3.4.1, is used to get and/or set values for particular VISA attributes.

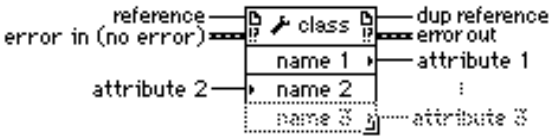


Figure 3.4.1. Property Node

3.4.1 Attributes

Table 3.4.1 lists the G Language binding and access privilege for each VISA attribute.

Table 3.4.1. Attributes

Attribute Name	G Language Binding	Access Privilege
VI_ATTR_RSRC_NAME	Resource Name	Read Only
VI_ATTR_RSRC_IMPL_VERSION	Version of Implementation	Read Only
VI_ATTR_RSRC_LOCK_STATE	Resource Lock State	Read Only
VI_ATTR_MAX_QUEUE_LENGTH	Maximum Queue Length	Read/Write
VI_ATTR_USER_DATA	User Data	Read/Write
VI_ATTR_FDC_CHNL	Fast Data Channel: Channel Number	Read/Write
VI_ATTR_FDC_MODE	Fast Data Channel: Channel Mode	Read/Write
VI_ATTR_FDC_GEN_SIGNAL_EN	Fast Data Channel: Signal Enable	Read/Write
VI_ATTR_FDC_USE_PAIR	Fast Data Channel: Use Channel Pairs	Read/Write
VI_ATTR_SEND_END_EN	Send End Enable	Read/Write
VI_ATTR_TERMCHAR	Termination Character	Read/Write
VI_ATTR_TMO_VALUE	Timeout Value	Read/Write
VI_ATTR_IO_PROT	IO Protocol	Read/Write
VI_ATTR_ASRL_BAUD	Baud Rate	Read/Write
VI_ATTR_ASRL_DATA_BITS	Data Bits	Read/Write
VI_ATTR_ASRL_PARITY	Parity	Read/Write
VI_ATTR_ASRL_STOP_BITS	Stop Bits	Read/Write
VI_ATTR_ASRL_FLOW_CNTRL	Flow Control	Read/Write
VI_ATTR_SUPPRESS_END_EN	Suppress End Enable	Read/Write
VI_ATTR_TERMCHAR_EN	Termination Character Enable	Read/Write
VI_ATTR_SRC_INCREMENT	Source Increment Count	Read/Write
VI_ATTR_DEST_INCREMENT	Destination Increment Count	Read/Write
VI_ATTR_CMDR_LA	VXI Commander Logical Address	Read Only
VI_ATTR_MAINFRAME_LA	Mainframe Logical Address	Read Only
VI_ATTR_WIN_BASE_ADDR	Window Base Address	Read Only
VI_ATTR_WIN_SIZE	Window Size	Read Only
VI_ATTR_ASRL_AVAIL_NUM	Number of Bytes at Serial Port	Read Only
VI_ATTR_MEM_BASE	VXI Memory Base Address	Read Only
VI_ATTR_ASRL_END_IN	End Mode for Reads	Read/Write
VI_ATTR_ASRL_END_OUT	End Mode for Writes	Read/Write
VI_ATTR_INTR_STATUS_ID	Interrupt Status ID	Read Only
VI_ATTR_RECV_INTR_LEVEL	Received Interrupt Level	Read Only
VI_ATTR_VXI_LA	VXI Logical Address	Read Only

(continues)

Table 3.4.1 Attributes (Continued)

Attribute Name	G Language Binding	Access Privilege
VI_ATTR_WIN_ACCESS	Window Access	Read Only
VI_ATTR_MEM_SPACE	VXI Memory Address Space	Read Only
VI_ATTR_MODEL_CODE	Model Code	Read Only
VI_ATTR_SLOT	Slot	Read Only
VI_ATTR_IMMEDIATE_SERV	Immediate Servant	Read Only
VI_ATTR_INTF_PARENT_NUM	Interface Number of Parent	Read Only
VI_ATTR_RSRC_SPEC_VERSION	Version of Specification	Read Only
VI_ATTR_INTF_TYPE	Interface Type	Read Only
VI_ATTR_GPIB_PRIMARY_ADDR	Primary Address	Read Only
VI_ATTR_GPIB_SECONDARY_ADDR	Secondary Address	Read Only
VI_ATTR_RSRC_MANF_NAME	Resource Manufacturer Name	Read Only
VI_ATTR_RSRC_MANF_ID	Resource Manufacturer Identification	Read Only
VI_ATTR_INTF_NUM	Interface Number	Read Only
VI_ATTR_TRIG_ID	Trigger Identifier	Read/Write
VI_ATTR_INTF_INST_NAME	Interface Description	Read Only
VI_ATTR_GPIB_READDR_EN	Readdressing	Read/Write
VI_ATTR_GPIB_UNADDR_EN	Unaddressing	Read/Write
VI_ATTR_SRC_ACCESS_PRIV	Source Access Privilege	Read/Write
VI_ATTR_DEST_ACCESS_PRIV	Destination Access Privilege	Read/Write
VI_ATTR_WIN_ACCESS_PRIV	Window Access Privilege	Read/Write
VI_ATTR_SRC_BYTE_ORDER	Source Byte Order	Read/Write
VI_ATTR_DEST_BYTE_ORDER	Destination Byte Order	Read/Write
VI_ATTR_WIN_BYTE_ORDER	Window Byte Order	Read/Write
VI_ATTR_ASRL_CTS_STATE	Modem Line Settings:Line CTS State	Read Only
VI_ATTR_ASRL_DCD_STATE	Modem Line Settings:Line DCD State	Read Only
VI_ATTR_ASRL_DSR_STATE	Modem Line Settings:Line DSR State	Read Only
VI_ATTR_ASRL_DTR_STATE	Modem Line Settings:Line DTR State	Read/Write
VI_ATTR_ASRL_RI_STATE	Modem Line Settings: Line RI State	Read Only
VI_ATTR_ASRL_RTS_STATE	Modem Line Settings:Line RTS State	Read/Write
VI_ATTR_EVENT_TYPE	Event Type	Read Only
VI_ATTR_SIGP_STATUS_ID	Signal Processor Status ID	Read Only
VI_ATTR_RECV_TRIG_ID	Received Trigger ID	Read Only
VI_ATTR_MANF_ID	Manufacturer Identification	Read Only
VI_ATTR_MEM_SIZE	VXI Memory Size	Read Only
VI_ATTR_ASRL_REPLACE_CHAR	Error Replacement Character	Read/Write

(continues)

Table 3.4.1 Attributes (Continued)

Attribute Name	G Language Binding	Access Privilege
VI_ATTR_ASRL_XON_CHAR	Flow Control XON Character	Read/Write
VI_ATTR_ASRL_XOFF_CHAR	Flow Control XOFF Character	Read/Write
VI_ATTR_GPIB_REN_STATE	Line REN State	Read Only
VI_ATTR_DMA_ALLOW_EN	Allow DMA Transfers	Read/Write
VI_ATTR_GPIB_ATN_STATE	Line ATN State	Read Only
VI_ATTR_GPIB_ADDR_STATE	Addressed State	Read Only
VI_ATTR_GPIB_CIC_STATE	CIC State	Read Only
VI_ATTR_GPIB_NDAC_STATE	Line NDAC State	Read Only
VI_ATTR_GPIB_SRQ_STATE	Line SRQ State	Read Only
VI_ATTR_GPIB_SYS_CNTRL_STATE	System Controller State	Read/Write
VI_ATTR_GPIB_HS488_CBL_LEN	HS488 Cable Length	Read/Write
VI_ATTR_VXI_DEV_CLASS	VXI Device Class	Read Only
VI_ATTR_MANF_NAME	Manufacturer Name	Read Only
VI_ATTR_MODEL_NAME	Model Name	Read Only
VI_ATTR_VXI_VME_INTR_STATUS	Asserted VXI/VME Interrupt Lines	Read Only
VI_ATTR_VXI_TRIG_STATUS	Asserted VXI Trigger Lines	Read Only
VI_ATTR_VXI_VME_SYSFAIL_STATE	VXI/VME System Failure State	Read Only
VI_ATTR_DEV_STATUS_BYTE	Device Status Byte	Read/Write
VI_ATTR_FILE_APPEND_EN	File Append Enable	Read/Write
VI_ATTR_VXI_TRIG_SUPPORT	Supported VXI Trigger Lines	Read Only
VI_ATTR_TCPIP_ADDR	Dot-Notation Address	Read Only
VI_ATTR_TCPIP_HOSTNAME	Computer Hostname	Read Only
VI_ATTR_TCPIP_PORT	Port Number	Read Only
VI_ATTR_GPIB_RECV_CIC_STATE	Received CIC State	Read Only
VI_ATTR_RSRC_CLASS	Resource Class	Read Only
VI_ATTR_TCPIP_DEVICE_NAME	LAN Device Name	Read Only
VI_ATTR_TCPIP_NODELAY	No Packet Delay	Read/Write
VI_ATTR_TCPIP_KEEPALIVE	Keep-Alive Packets	Read/Write
VI_ATTR_RECV_TCPIP_ADDR	Received TCP/IP Address	Read Only
VI_ATTR_4882_COMPLIANT	Is 488.2 Compliant	Read Only
VI_ATTR_USB_SERIAL_NUM	Serial Number	Read Only
VI_ATTR_USB_INTFC_NUM	USB Interface Number	Read Only
VI_ATTR_USB_PROTOCOL	USB Protocol	Read Only
VI_ATTR_USB_MAX_INTR_SIZE	Maximum Interrupt Size	Read/Write
VI_ATTR_USB_RECV_INTR_SIZE	USB Received Interrupt Size	Read Only

(continues)

Table 3.4.1 Attributes (Continued)

VI_ATTR_PXI_DEV_NUM	PCI Device Number	Read Only
VI_ATTR_PXI_FUNC_NUM	PCI Function Number	Read Only
VI_ATTR_PXI_BUS_NUM	PCI Bus Number	Read Only
VI_ATTR_PXI_CHASSIS	PXI Chassis Number	Read Only
VI_ATTR_PXI_SLOTPATH	Slot Path	Read Only
VI_ATTR_PXI_SLOT_LBUS_LEFT	Slot Local Bus Left	Read Only
VI_ATTR_PXI_SLOT_LBUS_RIGHT	Slot Local Bus Right	Read Only
VI_ATTR_PXI_TRIG_BUS	Trigger Bus Number	Read Only
VI_ATTR_PXI_STAR_TRIG_BUS	Star Trigger Bus Number	Read Only
VI_ATTR_PXI_STAR_TRIG_LINE	Star Trigger Line	Read Only
VI_ATTR_PXI_MEM_TYPE_BAR0	PCI Resources:BAR0 Address Type	Read Only
VI_ATTR_PXI_MEM_TYPE_BAR1	PCI Resources:BAR1 Address Type	Read Only
VI_ATTR_PXI_MEM_TYPE_BAR2	PCI Resources:BAR2 Address Type	Read Only
VI_ATTR_PXI_MEM_TYPE_BAR3	PCI Resources:BAR3 Address Type	Read Only
VI_ATTR_PXI_MEM_TYPE_BAR4	PCI Resources:BAR4 Address Type	Read Only
VI_ATTR_PXI_MEM_TYPE_BAR5	PCI Resources:BAR5 Address Type	Read Only
VI_ATTR_PXI_MEM_BASE_BAR0	PCI Resources:BAR0 Address Base	Read Only
VI_ATTR_PXI_MEM_BASE_BAR1	PCI Resources:BAR1 Address Base	Read Only
VI_ATTR_PXI_MEM_BASE_BAR2	PCI Resources:BAR2 Address Base	Read Only
VI_ATTR_PXI_MEM_BASE_BAR3	PCI Resources:BAR3 Address Base	Read Only
VI_ATTR_PXI_MEM_BASE_BAR4	PCI Resources:BAR4 Address Base	Read Only
VI_ATTR_PXI_MEM_BASE_BAR5	PCI Resources:BAR5 Address Base	Read Only
VI_ATTR_PXI_MEM_SIZE_BAR0	PCI Resources:BAR0 Address Size	Read Only
VI_ATTR_PXI_MEM_SIZE_BAR1	PCI Resources:BAR1 Address Size	Read Only
VI_ATTR_PXI_MEM_SIZE_BAR2	PCI Resources:BAR2 Address Size	Read Only
VI_ATTR_PXI_MEM_SIZE_BAR3	PCI Resources:BAR3 Address Size	Read Only
VI_ATTR_PXI_MEM_SIZE_BAR4	PCI Resources:BAR4 Address Size	Read Only
VI_ATTR_PXI_MEM_SIZE_BAR5	PCI Resources:BAR5 Address Size	Read Only
VI_ATTR_STATUS	Status	Read Only
VI_ATTR_PXI_IS_EXPRESS	Express Settings:Is PCI Express	Read Only
VI_ATTR_PXI_SLOT_LWIDTH	Express Settings:Slot Link Width	Read Only
VI_ATTR_PXI_MAX_LWIDTH	Express Settings:Maximum Link Width	Read Only
VI_ATTR_PXI_ACTUAL_LWIDTH	Express Settings:Actual Link Width	Read Only
VI_ATTR_PXI_DSTAR_BUS	Express Settings:D-Star Bus Number	Read Only
VI_ATTR_PXI_DSTAR_SET	Express Settings:D-Star Set	Read Only
VI_ATTR_TCPIP_HISLIP_OVERLAP_EN	Overlapped Message Mode Enabled	Read/Write
VI_ATTR_TCPIP_HISLIP_VERSION	HiSLIP Protocol Version On Server	Read Only
VI_ATTR_TCPIP_HISLIP_MAX_MESSAGE_KB	Maximum HiSLIP Message Size	Read/Write

RULE 3.4.1

All attribute operations specified in Section 3.4 **SHALL** be implemented as specified.

3.5 Event Type Values

Table 3.5.1 shows the event type values used for all G-based framework bindings.

Table 3.5.1. Event Type Values

Attribute Names	Values
VI_EVENT_IO_COMPLETION	3FFF2009h
VI_EVENT_TRIG	BFFF200Ah
VI_EVENT_SERVICE_REQ	3FFF200Bh
VI_EVENT_CLEAR	3FFF200Dh
VI_EVENT_GPIB_CIC	3FFF2012h
VI_EVENT_GPIB_TALK	3FFF2013h
VI_EVENT_GPIB_LISTEN	3FFF2014h
VI_EVENT_VXI_VME_SYSFAIL	3FFF201Dh
VI_EVENT_VXI_VME_SYSRESET	3FFF201Eh
VI_EVENT_VXI_SIGP	3FFF2020h
VI_EVENT_VXI_VME_INTR	BFFF2021h
VI_EVENT_TCPIP_CONNECT	3FFF2036h
VI_EVENT_USB_INTR	3FFF2037h
VI_EVENT_PXI_INTR	3FFF2022h
VI_ALL_ENABLED_EVENTS	3FFF7FFFh

Note: ***The VI_EVENT_EXCEPTION event is not supported in the G-based frameworks for two reasons:***

- ***Exceptions require callback support.***
- ***The G-based frameworks already provide an exception data flow mechanism using error clusters.***

3.6 Values and Ranges

Table 3.6.1 shows the values used in all G-based framework bindings.

Table 3.6.1. Values and Ranges

Name	Value
VI_FIND_BUFLEN	256
VI_NULL	0
VI_TRUE	1
VI_FALSE	0
VI_INTF_GPIB	1
VI_INTF_VXI	2
VI_INTF_GPIB_VXI	3
VI_INTF_ASRL	4
VI_INTF_TCPIP	6
VI_NORMAL	1
VI_FDC	2
VI_HS488	3
VI_ASRL488	4
VI_FDC_NORMAL	1
VI_FDC_STREAM	2
VI_A16_SPACE	1
VI_A24_SPACE	2
VI_A32_SPACE	3
VI_UNKNOWN_SLOT	-1
VI_UNKNOWN_LA	-1
VI_UNKNOWN_LEVEL	-1
VI_QUEUE	1
VI_HNDLR	2
VI_SUSPEND_HNDLR	4
VI_ALL_MECH	FFFFh
VI_ANY_HNDLR	0
VI_TRIG_SW	-1
VI_TRIG_TTL0	0
VI_TRIG_TTL1	1
VI_TRIG_TTL2	2

Name	Value
VI_TRIG_TTL5	5
VI_TRIG_TTL6	6
VI_TRIG_TTL7	7
VI_TRIG_ECL0	8
VI_TRIG_ECL1	9
VI_TRIG_PANEL_IN	27
VI_TRIG_PANEL_OUT	28
VI_TRIG_PROT_DEFAULT	0
VI_TRIG_PROT_ON	1
VI_TRIG_PROT_OFF	2
VI_TRIG_PROT_SYNC	5
VI_READ_BUF	1
VI_WRITE_BUF	2
VI_READ_BUF_DISCARD	4
VI_WRITE_BUF_DISCARD	8
VI_ASRL_IN_BUF	16
VI_ASRL_OUT_BUF	32
VI_ASRL_IN_BUF_DISCARD	64
VI_ASRL_OUT_BUF_DISCARD	128
VI_FLUSH_ON_ACCESS	1
VI_FLUSH_WHEN_FULL	2
VI_FLUSH_DISABLE	3
VI_NMAPPED	1
VI_USE_OPERS	2
VI_DEREF_ADDR	3
VI_TMO_IMMEDIATE	0
VI_TMO_INFINITE	FFFFFFFFh
VI_NO_LOCK	0
VI_EXCLUSIVE_LOCK	1
VI_SHARED_LOCK	2
VI_NO_SEC_ADDR	FFFFh

(continues)

Table 3.6.1. Values and Ranges (Continued)

Name	Value
VI_TRIG_TTL3	3
VI_TRIG_TTL4	4
VI_ASRL_PAR_EVEN	2
VI_ASRL_PAR_MARK	3
VI_ASRL_PAR_SPACE	4
VI_ASRL_STOP_ONE	10
VI_ASRL_STOP_ONE5	15
VI_ASRL_STOP_TWO	20
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_END_NONE	0
VI_ASRL_END_LAST_BIT	1
VI_ASRL_END_TERMCHAR	2
VI_ASRL_END_BREAK	3
VI_BIG_ENDIAN	0
VI_LITTLE_ENDIAN	1
VI_WIDTH_8	1
VI_WIDTH_16	2
VI_WIDTH_32	4
VI_STATE_ASSERTED	1
VI_STATE_UNASSERTED	0
VI_STATE_UNKNOWN	-1
VI_LOAD_CONFIG	4
VI_GPIB_HS488_DISABLED	0
VI_GPIB_HS488_NIMPL	-1
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
VI_UTIL_ASSERT_SYSRESET	1
VI_UTIL_ASSERT_SYSFAIL	2
VI_UTIL_DEASSERT_SYSFAIL	3

Name	Value
VI_ASRL_PAR_NONE	0
VI_ASRL_PAR_ODD	1
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_LOCAL_SPACE	0
VI_GPIB_REN_DEASSERT	0
VI_GPIB_REN_ASSERT	1
VI_GPIB_REN_DEASSERT_GTL	2
VI_GPIB_REN_ASSERT_ADDRESS	3
VI_GPIB_REN_ASSERT_LLO	4
VI_GPIB_REN_ASSERT_ADDRESS_LLO	5
VI_GPIB_REN_ADDRESS_GTL	6
VI_VXI_CMD16	0200h
VI_VXI_CMD16_RESP16	0202h
VI_VXI_RESP16	0002h
VI_VXI_CMD32	0400h
VI_VXI_CMD32_RESP16	0402h
VI_VXI_CMD32_RESP32	0404h
VI_VXI_RESP32	0004h
VI_GPIB_ATN_DEASSERT	0
VI_GPIB_ATN_ASSERT	1
VI_GPIB_ATN_DEASSERT_HANDSHAKE	2
VI_GPIB_ATN_ASSERT_IMMEDIATE	3
VI_ASSERT_SIGNAL	-1
VI_ASSERT_USE_ASSIGNED	0
VI_ASSERT_IRQ1	1
VI_ASSERT_IRQ2	2
VI_ASSERT_IRQ3	3
VI_ASSERT_IRQ4	4

(continues)

Table 3.6.1. Values and Ranges (Continued)

Name	Value
VI_TRIG_ALL	-2
VI_ASSERT_IRQ7	7
VI_GPIB_UNADDRESSED	0
VI_GPIB_TALKER	1
VI_GPIB_LISTENER	2
VI_PROT_4882_STRS	4
VI_INTF_USB	7
VI_PROT_FDC	2
VI_OPAQUE_SPACE	FFFFh
VI_INTF_PXI	5
VI_PXI_ALLOC_SPACE	9
VI_PXI_CFG_SPACE	10
VI_PXI_BAR0_SPACE	11
VI_PXI_BAR1_SPACE	12
VI_PXI_BAR2_SPACE	13
VI_PXI_BAR3_SPACE	14
VI_PXI_BAR4_SPACE	15
VI_PXI_BAR5_SPACE	16
VI_PXI_ADDR_NONE	0
VI_PXI_ADDR_MEM	1
VI_PXI_ADDR_IO	2
VI_PXI_ADDR_CFG	3
VI_TRIG_ECL2	10
VI_TRIG_ECL3	11
VI_TRIG_ECL4	12
VI_TRIG_ECL5	13
VI_TRIG_STAR_SLOT1	14
VI_TRIG_STAR_SLOT2	15
VI_TRIG_STAR_SLOT3	16
VI_TRIG_STAR_SLOT4	17
VI_TRIG_STAR_SLOT5	18
VI_TRIG_STAR_SLOT6	19
VI_TRIG_STAR_SLOT7	20
VI_TRIG_STAR_SLOT8	21

Name	Value
VI_ASSERT_IRQ5	5
VI_ASSERT_IRQ6	6
VI_IO_IN_BUF	16
VI_IO_OUT_BUF	32
VI_IO_IN_BUF_DISCARD	64
VI_IO_OUT_BUF_DISCARD	128
VI_PROT_NORMAL	1
VI_PROT_HS488	3
VI_PROT_USBTMC_VENDOR	5
VI_UNKNOWN_CHASSIS	-1
VI_UNKNOWN_TRIG	-1
VI_PXI_LBUS_STAR_TRIG_BUS_0	1000
VI_PXI_LBUS_STAR_TRIG_BUS_1	1001
VI_PXI_LBUS_STAR_TRIG_BUS_2	1002
VI_PXI_LBUS_STAR_TRIG_BUS_3	1003
VI_PXI_LBUS_STAR_TRIG_BUS_4	1004
VI_PXI_LBUS_STAR_TRIG_BUS_5	1005
VI_PXI_LBUS_STAR_TRIG_BUS_6	1006
VI_PXI_LBUS_STAR_TRIG_BUS_7	1007
VI_PXI_LBUS_STAR_TRIG_BUS_8	1008
VI_PXI_LBUS_STAR_TRIG_BUS_9	1009
VI_PXI_STAR_TRIG_CONTROLLER	1413
VI_TRIG_PROT_RESERVE	6
VI_TRIG_PROT_UNRESERVE	7
VI_TRIG_STAR_SLOT9	22
VI_TRIG_STAR_SLOT10	23
VI_TRIG_STAR_SLOT11	24
VI_TRIG_STAR_SLOT12	25
VI_TRIG_STAR_INSTR	26
VI_TRIG_STAR_VXI0	29
VI_TRIG_STAR_VXI1	30
VI_TRIG_STAR_VXI2	31
VI_TRIG_TTL8	32
VI_TRIG_TTL9	33
VI_TRIG_TTL10	34
VI_TRIG_TTL11	35

RULE 3.6.1

The range of the attribute VI_ATTR_USER_DATA **SHALL** be 0 to FFFFFFFFh.