VISA for PXI Specification

PCI eXtensions for Instrumentation

An Implementation of ***CompactPCI™***

PXI-3
Revision 1.0
September 25, 2003

IMPORTANT INFORMATION

Copyright

NOTICE

The *VISA for PXI Specification* is authored and copyrighted by the PXI Systems Alliance. The intent of the PXI Systems Alliance is for the *VISA for PXI Specification* to be an open industry standard supported by a wide variety of vendors and products. Vendors and users who are interested in developing PXI-compatible products or services, as well as parties who are interested in working with the PXI Systems Alliance to further promote PXI as an open industry standard, are invited to contact the PXI Systems Alliance for further information.

The PXI Systems Alliance wants to receive your comments on this specification. Visit the PXI Systems Alliance Web site at `http://www.pxisa.org/` for contact information and to learn more about the PXI Systems Alliance.

The attention of adopters is directed to the possibility that compliance with or adoption of the PXI Systems Alliance specifications may require use of an invention covered by patent rights. The PXI Systems Alliance shall not be responsible for identifying patents for which a license may be required by any PXI Systems Alliance specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. PXI Systems Alliance specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

The information contained in this document is subject to change without notice. The material in this document details a PXI Systems Alliance specification in accordance with the license and notices set forth on this page. This document does not represent a commitment to implement any portion of this specification in any company's products.

The PXI Systems Alliance makes 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 PXI Systems Alliance 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.

Compliance with this specification does not absolve manufacturers of PXI equipment from the requirements of safety and regulatory agencies (UL, CSA, FCC, IEC, etc.).

Trademarks

PXI™ is a trademark of the PXI Systems Alliance.

PICMG™ and CompactPCI® are trademarks of the PCI Industrial Computation Manufacturers Group.

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

VISA for PXI Specification Revision History

This section is an overview of the revision history of the *VISA for PXI Specification*.

Revision 1.0, September 25, 2003

This is the first public revision of the *VISA for PXI Specification*.

Contents

Tables

1. Introduction

This section explains the objectives and scope of the VISA Library Extensions for PXI Systems. It also describes the intended audience and lists relevant terminology and documents. This specification is intended to describe an I/O API for accessing PXI modules, based on the VISA API. Refer to the *PXI Software Specification* for general background on the PXI software models, including the hardware description files used to identify and manage the resources in a PXI system.

1.1 Objectives

The *PXI Software Specification* was created to supplement the *PXI Hardware Specification* in clarifying and addressing common software requirements in PXI Systems. The *PXI Software Specification* provides a basis for interoperability among software components in a PXI system, based on configuration files. This *VISA for PXI Specification* extends the *PXI Software Specification* by defining a common API for performing low-level I/O operations to modules in a PXI system.

To promote software interoperability, section 3.4.3 and section 3.5 of version 2.1 of the *PXI Software Specification* recommend that all system controller modules should be supplied with a VISA implementation that supports the PXI bus. These sections require that, once a specification for VISA for PXI has been available for three months, all system controller modules are required to include an implementation. This *VISA for PXI Specification* is intended to fulfill that condition. Therefore, three months after the release of this specification, PXI controller modules that claim compliance with the *PXI Software Specification* will be required to ship with at VISA for PXI implementation that conforms to this specification.

1.2 Intended Audience and Scope

This specification is intended for developers of instrument drivers for PXI modules, so that those developers may rely on a common, binary-compatible API for performing I/O operations in a PXI system. This specification is also intended for developers of VISA-compliant I/O software to be included with PXI system controller modules.

The *VISA for PXI Specification* defines attributes, events, and methods in a VISA implementation. The VISA library, as defined by *VXIplug&play Specification VPP-4.3*, provides a common API for performing I/O operations on standard instrumentation buses, such as GPIB, VXI, Ethernet, and RS-232. The *VISA for PXI Specification* defines how a VISA library shall provide support for PXI.

The purpose of the *VISA for PXI Specification* is to provide API-level compatibility among I/O layers that implementers of instrument drivers or other module drivers can use. Note that the data bus in PXI is based on the *PCI Local Bus Specification*. PCI is a native bus on computer systems, managed directly by the operating system. A VISA implementation that supports PXI needs to register with the operating system its intention to manage certain devices or buses in a PXI system. That registration mechanism is described in section 2.4.3.

1.3 Background and Terminology

This section defines the acronyms and key words referred to throughout this specification. This specification uses the following acronyms:

- **API**—Application Programming Interface
- **CompactPCI** – PICMG 2.0 Specification
- **PCI**—Peripheral Component Interconnect; electrical specification defined by PCISIG
- **PCISIG**—PCI Special Interest Group
- **PICMG**—PCI Industrial Computer Manufacturers Group
- **PXI**—PCI eXtensions for Instrumentation

- **VISA**—Virtual Instrument Software Architecture
- **VPP**—VXI*plug&play* Specification

This specification uses several key words, which are defined as follows:

RULE: Rules SHALL be followed to ensure compatibility. A rule is characterized by the use of the words SHALL and SHALL NOT.

RECOMMENDATION: Recommendations consist of advice to implementers that will affect the usability of the final module. A recommendation is characterized by the use of the words SHOULD and SHOULD NOT.

PERMISSION: Permissions clarify the areas of the specification that are not specifically prohibited.

Permissions reassure the reader that a certain approach is acceptable and will cause no problems. A permission is characterized by the use of the word MAY.

OBSERVATION: 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.

MAY: A key word indicating flexibility of choice with no implied preference. This word is usually associated with a permission.

SHALL: A key word indicating a mandatory requirement. Designers SHALL implement such mandatory requirements to ensure interchangeability and to claim conformance with the specification. This word is usually associated with a rule.

SHOULD: A key word indicating flexibility of choice with a strongly preferred implementation. This word is usually associated with a recommendation.

1.4 Applicable Documents

This specification defines extensions to the base PCI and CompactPCI specifications referenced in this section. It is assumed that the reader has a thorough understanding of PCI and CompactPCI. The CompactPCI specification refers to several other applicable documents with which the reader may want to become familiar.

This specification refers to the following documents directly:

- *PXI-1: PXI Hardware Specification*, Rev 2.1
- *PXI-2: PXI Software Specification*, Rev 2.2
- *PXI-4: PXI Module Description File Specification*, Rev 1.0
- *PCI Local Bus Specification*, Rev. 2.2
- *PICMG 2.0 R3.0 CompactPCI Specification*
- *VPP-4.3: The VISA Library*, Rev 2.2
- *VPP-4.3.2: VISA Implementation Specification for Textual Languages*, Rev 2.2
- *VPP-4.3.3: VISA Implementation Specification for the G Language*, Rev 2.2
- *VPP-4.3.4: VISA Implementation Specification for COM*, Rev 1.0

1.5 Useful Web Sites

Below is a list of web site links that at the time of publication of this specification point to sites with information useful in the understanding and design of PXI products.

- `http://www.pxisa.org/` PXI specifications
- `http://www.picmg.org/` PICMG specifications
- `http://www.ieee.org/` IEEE specifications
- `http://www.iec.org/` IEC specifications
- `http://www.pcisig.com/` PCI specifications
- `http://www.vita.com/` VME specifications
- `http://www.vxi.org/` VXI specifications
- `http://www.vxipnp.org/` VISA specifications

2. PXI Extensions to the VISA Library

This section defines the extensions to the *VXIplug&play Systems Alliance* specification *VPP-4.3: The VISA Library* that are required to support PXI systems. Extension information is provided about each section of the *VPP-4.3* specification. Each of the subsections 2.1 through 2.6 of this section corresponds to sections 1 through 6 of the VISA Library specification, with each of the subsections herein providing extension information about the corresponding VISA Library specification section. Similarly, subsections 2.7 and 2.8 of this section provide extension information about *VPP-4.3* appendices A and B, respectively.

2.1 Introduction

As noted, the vision of the *VXIplug&play Systems Alliance* for a multivendor system architecture extends beyond the scope of the VXI specification. The VISA library has now been defined to include resources for VXI, GPIB, asynchronous serial buses, and TCP/IP (including VXI-11 and raw sockets). This specification adds PXI to the list of supported buses.

2.2 Overview of the VISA Library Specification

The VISA library specification is organized with bus-independent and interface-independent rules and definitions in the first four sections, followed by bus and resource-dependent material in the other two sections and appendices. The material presented in the first four sections of VPP-4.3 is applicable to all interfaces, including PXI.

2.3 VISA Resource Template

RULE: A VISA implementation for PXI must support all VISA Resource Template functionality defined in Section 3 of VPP-4.3.

2.4 VISA Resource Management

RULE: A VISA implementation for PXI must support all VISA Resource Management functionality defined in Section 4 of VPP-4.3.

2.4.1 Address String Grammar

In a VISA resource string, the PXI keyword establishes communication with a PXI system. The Address String grammar for PXI VISA resources is shown in Table 2-1.

Table 2-1. Address String Grammar

Description	Grammar
Bus/device/function string	PXI[*interface*]::*bus-device*[.*function*][::INSTR]
Bus/device/function legacy string	PXI[*bus*]::*device*[::*function*][::INSTR]
Memory access string	PXI[*interface*]::MEMACC
Chassis/slot string	PXI[*interface*]::CHASSIS*chassis*::SLOT*slot*[::FUNC*function*][::INSTR]

In these strings, the *interface* parameter refers to the interface to PXI.

RULE: In a system where all PCI devices are accessible through a single configuration address space, the *interface* parameter SHALL be zero (0) for all resources.

In these strings, the *bus*, *device*, and *function* parameters refer to the PCI bus number, PCI device number, and PCI function number that would be used to access the resource in PCI configuration space. The *chassis* and *slot* parameters correspond to the chassis number and slot number attributes of the resource.

Notice that the address string for a PXI INSTR resource has three acceptable formats.

RULE: A VISA implementation that supports PXI INSTR resources SHALL support the Bus/device/function string format.

RULE: A VISA implementation that supports PXI MEMACC resources SHALL support the Memory access string format.

RULE: A VISA implementation that supports PXI INSTR resources SHALL support the Bus/device/function legacy string format.

RULE: A VISA implementation that supports PXI INSTR resources SHALL support the Chassis/slot string format.

Table 2-2. Examples of PXI Address Strings

Address String	Description
PXI0::3-18::INSTR	PXI device 18 on bus 3.
PXI0::3-18.2::INSTR	Function 2 on PXI device 18 on bus 3.
PXI0::21::INSTR	PXI device 21 on bus 0.
PXI0::MEMACC	Access to system controller memory available to devices in the PXI system.
PXI0::CHASSIS1::SLOT4::INSTR	PXI device in slot 4 of chassis 1.

OBSERVATION: The VISA resource string for a single-function device on bus zero (0) is identical in both formats for PXI INSTR resources.

OBSERVATION: The Bus/device/function legacy string format does not allow for multiple PXI systems with separate address spaces. Although PCI-based systems typically have a single address space today, there may be a need for multiple address spaces in the future.

OBSERVATION: The operation `viFindRsrc()` allows a user of the VISA Library to find a VISA resource by matching values of attributes. Thus, using `viFindRsrc`, a user of the VISA library can find a resource that is in a specific chassis and slot by matching on `VI_ATTR_PXI_CHASSIS` and `VI_ATTR_SLOT`, or at a specific slot path, by matching on `VI_ATTR_PXI_SLOTPATH`.

2.4.2 Search Services

As noted earlier, a PXI INSTR resource may have multiple strings to identify it, to allow it to be identified by a bus/device/function address. The VISA search services, however, are intended to find one unique string for each resource. Therefore, a VISA implementation must return only one string for each VISA resource.

RULE: A VISA implementation that supports PXI INSTR resources SHALL match and return only one resource string per PXI INSTR resource.

RULE: VISA implementation that supports PXI INSTR SHALL be capable of returning the bus/device/function format for the string.

PERMISSION: A VISA implementation that supports PXI INSTR MAY provide configuration options to return other resource string formats for PXI resources, not limited to those defined in this specification, as long as only one resource string is returned per PXI resource.

OBSERVATION: Observation 4.4.7 of VPP 4-3 notes that the behavior of `viFindRsrc` in a VISA implementation is allowed to be modifiable through an optional, external configuration utility.

2.4.3 Module Registration

As noted earlier, a VISA implementation needs to register with the operating system its intention to manage certain devices or buses. A VISA implementation must provide this mechanism.

RULE: A VISA implementation that supports PXI INSTR resources SHALL provide a tool for registering modules using the `module.ini` files specified in the PXI Software Specification. The tool SHALL provide a mechanism for registering those devices in a programmatic or scriptable manner.

RECOMMENDATION: A VISA implementation that supports PXI INSTR resources should provide an interactive tool for registering modules that does not require a `module.ini` file.

OBSERVATION: End users will first install VISA, then use tools provided with the VISA implementation to register the module description file with the operating system, then install the hardware. For example, on Microsoft Windows operating systems VISA would read the module description and generate a Windows Setup Information (`.inf`) file that the operating system would then use to identify the hardware. Installing the software before the hardware ensures that the information in the module description file is available to the operating system when it needs to identify the hardware.

2.5 VISA Resource Classes

RULE: If a VISA implementation supports the PXI interface (`VI_INTF_PXI`), it shall implement the resource types INSTR and MEMACC.

2.5.1 PXI INSTR Resource

2.5.1.1 PXI INSTR Resource Attributes

OBSERVATION: The Generic INSTR Resource Attributes `VI_ATTR_IO_PROT`, `VI_ATTR_RD_BUF_OPER_MODE`, `VI_ATTR_SEND_END`, `VI_ATTR_SUPPRESS_END_EN`, `VI_ATTR_TERMCHAR`, `VI_ATTR_TERMCHAR_EN`, `VI_ATTR_WR_BUF_OPER_MODE`, and `VI_ATTR_FILE_APPEND_EN` are not defined for PXI resources. The VISA operations `viGetAttribute()` and `viSetAttribute()` return `VI_ERROR_NSUP_ATTR` for attributes that are not implemented for a VISA session, event, or find list.

The Generic INSTR resource attributes that are defined for PXI resources are `VI_ATTR_INTF_NUM`, `VI_ATTR_INTF_TYPE`, `VI_ATTR_INTF_INST_NAME`, `VI_ATTR_TMO_VALUE`, and `VI_ATTR_TRIG_ID`.

This table lists the PXI-specific attributes for INSTR resources.

Table 2-3. PXI-Specific INSTR Resource Attributes

Symbolic Name	Access Privilege		Data Type	Range
`VI_ATTR_PXI_BUS_NUM`	RO	Global	`ViUInt16`	0 to 255
`VI_ATTR_PXI_DEV_NUM`	RO	Global	`ViUInt16`	0 to 31
`VI_ATTR_PXI_FUNC_NUM`	RO	Global	`ViUInt16`	0 to 7

Table 2-3. PXI-Specific INSTR Resource Attributes (Continued)

Symbolic Name	Access Privilege		Data Type	Range
VI_ATTR_PXI_SLOTPATH	RO	Global	ViString	N/A
VI_ATTR_PXI_SLOT_LBUS_LEFT	RO	Global	ViInt16	0 to 32767 VI_UNKNOWN_SLOT
VI_ATTR_PXI_SLOT_LBUS_RIGHT	RO	Global	ViInt16	0 to 32767 VI_UNKNOWN_SLOT
VI_ATTR_PXI_TRIG_BUS	RO	Global	ViInt16	0 to 32767 VI_UNKNOWN_TRIG_BUS
VI_ATTR_PXI_STAR_TRIG_BUS	RO	Global	ViInt16	0 to 32767 VI_UNKNOWN_TRIG_BUS
VI_ATTR_PXI_STAR_TRIG_LINE	RO	Global	ViInt16	0 to 32767 VI_PXI_STAR_TRIG_LINE_UNKNOWN
VI_ATTR_PXI_MEM_TYPE_BAR*n* (where *n* is 0,1,2,3,4,5)	RO	Global	ViUInt16	VI_PXI_ADDR_MEM, VI_PXI_ADDR_IO, VI_PXI_ADDR_NONE
VI_ATTR_PXI_MEM_BASE_BAR*n* (where *n* is 0,1,2,3,4,5)	RO	Global	ViBusAddress	N/A
VI_ATTR_PXI_MEM_SIZE_BAR*n* (where *n* is 0,1,2,3,4,5)	RO	Global	ViBusSize	N/A
VI_ATTR_PXI_CHASSIS	RO	Global	ViInt16	0 to 255 VI_UNKNOWN_CHASSIS
VI_ATTR_SLOT	RO	Global	ViInt16	0 to 32767 VI_UNKNOWN_SLOT
VI_ATTR_MANF_ID	RO	Global	ViUInt16	0 to FFFFh
VI_ATTR_MODEL_CODE	RO	Global	ViUInt16	0 to FFFFh
VI_ATTR_MANF_NAME	RO	Global	ViString	N/A
VI_ATTR_MODEL_NAME	RO	Global	ViString	N/A
VI_ATTR_SRC_INCREMENT	R/W	Local	ViInt32	0 to 1
VI_ATTR_DEST_INCREMENT	R/W	Local	ViInt32	0 to 1
VI_ATTR_WIN_ACCESS	RO	Local	ViUInt16	VI_NMAPPED VI_USE_OPERS VI_DEREF_ADDR
VI_ATTR_WIN_BASE_ADDR	RO	Local	ViBusAddress	N/A
VI_ATTR_WIN_SIZE	RO	Local	ViBusSize	N/A

VI_ATTR_PXI_BUS_NUM — PCI bus number of this device.

VI_ATTR_PXI_DEV_NUM — PCI device number of this device.

VI_ATTR_PXI_FUNC_NUM	PCI function number of the device. All devices have a function 0. Multifunction devices will also support other function numbers.
VI_ATTR_PXI_SLOTPATH	Slot path of this device. PXI slot paths are a sequence of values representing the PCI device number and function number of a PCI module and each parent PCI bridge that routes the module to the host PCI bridge. The string format of the attribute value is device1[.function1][,device2[.function2]][,...].
VI_ATTR_PXI_SLOT_LBUS_LEFT	Slot number or special feature connected to the local bus left lines of this device.
VI_ATTR_PXI_SLOT_LBUS_RIGHT	Slot number or special feature connected to the local bus right lines of this device.
VI_ATTR_PXI_TRIG_BUS	Number of the trigger bus connected to this device in the chassis.
VI_ATTR_PXI_STAR_TRIG_BUS	Number of the star trigger bus connected to this device in the chassis.
VI_ATTR_PXI_STAR_TRIG_LINE	PXI_STAR line connected to this device.
VI_ATTR_PXI_MEM_TYPE_BAR*n*	Memory type (memory mapped or I/O mapped) used by the device in the specified BAR.
VI_ATTR_PXI_MEM_BASE_BAR*n*	Memory base address assigned to the specified BAR for this device.
VI_ATTR_PXI_MEM_SIZE_BAR*n*	Size of the memory assigned to the specified BAR for this device.
VI_ATTR_PXI_CHASSIS	Chassis number in which this device is located.
VI_ATTR_SLOT	Physical slot location of the device. If the slot number is not known, VI_UNKNOWN_SLOT is returned.
VI_ATTR_MANF_ID	Manufacturer identification number of the device. If Subsystem ID and Subsystem Vendor ID are defined for the device, then this attribute value is the Subsystem Vendor ID, else this attribute value is the PCI Vendor ID.
VI_ATTR_MODEL_CODE	Model code for the device. If Subsystem ID and Subsystem Vendor ID are defined for the device, then this attribute value is the Subsystem ID, else this attribute value is the PCI Device ID.
VI_ATTR_MANF_NAME	This string attribute is the manufacturerís 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.
VI_ATTR_MODEL_NAME	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.
VI_ATTR_SRC_INCREMENT	This is used in the viMoveIn*XX*() operation to specify how much 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 viMoveIn*XX*() operation moves from consecutive elements. If this attribute is set to 0, the viMoveIn*XX*() operation will always read from the same element, essentially treating the source as a FIFO register.
VI_ATTR_DEST_INCREMENT	This is used in the viMoveOut*XX*() operation to specify how much 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`*`()` operation moves into consecutive elements. If this attribute is set to 0, the `viMoveOut`*`XX`*`()` operation will always write to the same element, essentially treating the destination as a FIFO register.
`VI_ATTR_WIN_ACCESS`	Modes in which the current window may be accessed: not currently mapped, through operations `viPeek`*`XX`*`()` and `viPoke`*`XX`*`()` only, or through operations and/or by directly dereferencing the address parameter as a pointer.
`VI_ATTR_WIN_BASE_ADDR`	Base address of the interface bus to which this window is mapped.
`VI_ATTR_WIN_SIZE`	Size of the region mapped to this window.

2.5.1.2 PXI INSTR Resource Events

There is one PXI-specific event defined for PXI INSTR resources, `VI_EVENT_PXI_INTR`.

VI_EVENT_PXI_INTR

Description

Notification that a PCI Interrupt was received from the device.

Event Attribute

Symbolic Name	Access Privilege	Data Type	Range
`VI_ATTR_EVENT_TYPE`	RO	`ViEventType`	`VI_EVENT_PXI_INTR`

RULE: An INSTR resource implementation for a PXI system SHALL support the generation of the event `VI_EVENT_PXI_INTR`.

RULE: On some operating systems, it may be a requirement to handle PXI interrupts in the OS kernel environment. VISA implementations on such operating systems SHALL provide a mechanism for performing device-specific operations in the kernel in response to an interrupt. The PXI Module Description File Specification specifies a VISA Registration Descriptor for this purpose. This mechanism allows the event to be delivered to the instrument driver software in the application environment once the PXI interrupt has been safely removed in the OS kernel environment.

To implement the above rule, a VISA implementation could implement the following behavior.

1. The user, integrator, or instrument driver developer registers information from the module description file with the VISA implementation. The information about the device registered includes a description of these operations:
   a. How to detect whether the device is asserting a PXI interrupt (Operation DETECT).
   b. How to stop the device from asserting its PXI interrupt line. (Operation QUIESCE).
2. When the user enables events from the device, the VISA implementation reads the device description to find descriptions of the above operations.
3. Upon receiving an interrupt, the VISA implementation uses OS services combined with the DETECT operation on each device to determine which device is interrupting.
4. The VISA implementation uses the QUIESCE operation on the interrupting device.

5. The VISA implementation delivers the VI_EVENT_PXI_INTR to each session enabled for interrupts to that device.

Note In any implementation, the VISA client code must ensure that the device is enabled to drive the interrupt line again after handling the condition that caused the interrupt.

2.5.1.3 PXI INSTR Resource Operations

RULE: An INSTR resource implementation for a PXI system SHALL support the operations viAssertTrigger(), viIn8(), viIn16(), viIn32(), viOut8(), viOut16(), viOut32(), viMoveIn8(), viMoveIn16(), viMoveIn32(), viMoveOut8(), viMoveOut16(), viMoveOut32(), viMove(), viMoveAsync(), viMapAddress(), viUnmapAddress(), viPeek8(), viPeek16(), viPeek32(), viPoke8(), viPoke16(), and viPoke32().

OBSERVATION: The operations listed in the above rule and their behaviors are defined in Section 6 of VPP 4-3.

2.5.2 PXI MEMACC Resource

The PXI MEMACC resource provides the attributes, events, and operations necessary to allow controller software and physical PXI devices to access the memory on the PXI system controller. The PXI MEMACC resource does not necessarily provide unimpeded access to the entire memory space of the PXI bus.

PERMISSION: A VISA implementation that supports the PXI MEMACC resource may limit accesses to that resource to permit only accesses to memory allocated by viMemAlloc().

2.5.2.1 PXI MEMACC Resource Attributes

This table lists the PXI specific attributes for MEMACC resources.

Table 2-4. PXI-Specific MEMACC Resource Attributes

Symbolic Name	Access Privilege		Data Type	Range
VI_ATTR_WIN_ACCESS	RO	Local	ViUInt16	VI_NMAPPED VI_USE_OPERS VI_DEREF_ADDR
VI_ATTR_WIN_BASE_ADDR	RO	Local	ViBusAddress	N/A
VI_ATTR_WIN_SIZE	RO	Local	ViBusSize	N/A

2.5.2.2 PXI MEMACC Resource Events

OBSERVATION: The MEMACC Resource Events defined in Section 5.2.3 of the *VPP-4.3* are not interface dependent and apply equally to PXI. A MEMACC resource implementation for a PXI system is required to support the events listed that section.

2.5.2.3 PXI MEMACC Resource Operations

OBSERVATION: The MEMACC Resource Operations defined in Section 5.2.4 of the *VPP-4.3* are not interface dependent and apply equally to PXI. A MEMACC resource implementation for a PXI system is required to support the operations listed that section.

RULE: A MEMACC resource implementation for a PXI system SHALL support the operations `viMemAlloc()` and `viMemFree()`.

2.6 VISA Resource Specific Operations

2.6.1 PXI INSTR Resource Operations

2.6.1.1 Basic I/O Services

`viAssertTrigger(`**`vi, protocol`**`)`

For a PXI resource, `viAssertTrigger()` will reserve a trigger line for assertion, or release such a reservation. Instrument drivers should use `viAssertTrigger()` to ensure that they have ownership of a trigger line before performing any operation that could drive a signal onto that trigger line. The **`protocol`** parameter can be either `VI_TRIG_PROT_RESERVE` or `VI_TRIG_PROT_UNRESERVE`, which reserve a trigger line and release the reservation, respectively.

2.6.1.2 Memory I/O Services

All operations on a PXI INSTR resource that accept a **`space`** parameter to indicate the address space for bus access SHALL accept the following values for the **`space`** parameter: `VI_PXI_CFG_SPACE`, `VI_PXI_BAR0_SPACE`, `VI_PXI_BAR1_SPACE`, `VI_PXI_BAR2_SPACE`, `VI_PXI_BAR3_SPACE`, `VI_PXI_BAR4_SPACE`, `VI_PXI_BAR5_SPACE`.

All operations on a PXI MEMACC resource that accept a **`space`** parameter to indicate the address space for bus access SHALL accept the following value for the **`space`** parameter: `VI_PXI_ALLOC_SPACE`.

RULE: The **`offset`** parameter in the `viMemAlloc()` and `viMemFree()` operations on a PXI MEMACC resource SHALL be an absolute physical PCI address.

2.7 Required Attributes

This section lists the required attributes along with the range and default value of every resource described in this document.

Table 2-5. PXI-Specific INSTR Resource Attributes

Symbolic Name	Range	Default
`VI_ATTR_PXI_DEV_NUM`	0 to 31	N/A
`VI_ATTR_PXI_FUNC_NUM`	0 to 7	N/A
`VI_ATTR_PXI_BUS_NUM`	0 to 255	N/A
`VI_ATTR_PXI_CHASSIS`	0 to 255 `VI_UNKNOWN_CHASSIS`	N/A
`VI_ATTR_PXI_SLOTPATH`	N/A	N/A
`VI_ATTR_PXI_SLOT_LBUS_LEFT`	0 to 32767 `VI_UNKNOWN_SLOT`	N/A
`VI_ATTR_PXI_SLOT_LBUS_RIGHT`	0 to 32767 `VI_UNKNOWN_SLOT`	N/A
`VI_ATTR_PXI_TRIG_BUS`	0 to 32767 `VI_UNKNOWN_TRIG_BUS`	N/A

Table 2-5. PXI-Specific INSTR Resource Attributes (Continued)

Symbolic Name	Range	Default
VI_ATTR_PXI_STAR_TRIG_BUS	0 to 32767 VI_UNKNOWN_TRIG_BUS	N/A
VI_ATTR_PXI_STAR_TRIG_LINE	0 to 32767 VI_PXI_STAR_TRIG_LINE_ UNKNOWN	N/A
VI_ATTR_PXI_MEM_TYPE_BAR*n* (where n is 0,1,2,3,4,5)	VI_PXI_ADDR_MEM, VI_PXI_ADDR_IO, VI_PXI_ADDR_NONE	N/A
VI_ATTR_PXI_MEM_BASE_BAR*n* (where n is 0,1,2,3,4,5)	N/A	N/A
VI_ATTR_PXI_MEM_SIZE_BAR*n* (where n is 0,1,2,3,4,5)	N/A	N/A
VI_ATTR_SLOT	0 to 32767 VI_UNKNOWN_SLOT	N/A
VI_ATTR_MANF_ID	0 to FFFFh	N/A
VI_ATTR_MODEL_CODE	0 to FFFFh	N/A
VI_ATTR_MANF_NAME	N/A	N/A
VI_ATTR_MODEL_NAME	N/A	N/A
VI_ATTR_SRC_INCREMENT	0 to 1	1
VI_ATTR_DEST_INCREMENT	0 to 1	1
VI_ATTR_WIN_ACCESS	VI_NMAPPED VI_USE_OPERS VI_DEREF_ADDR	VI_NMAPPED
VI_ATTR_WIN_BASE_ADDR	N/A	N/A
VI_ATTR_WIN_SIZE	N/A	N/A

Table 2-6. PXI-Specific MEMACC Resource Attributes

Symbolic Name	Range	Default
VI_ATTR_SRC_INCREMENT	0 to 1	1
VI_ATTR_DEST_INCREMENT	0 to 1	1
VI_ATTR_WIN_ACCESS	VI_NMAPPED VI_USE_OPERS VI_DEREF_ADDR	VI_NMAPPED
VI_ATTR_WIN_BASE_ADDR	N/A	N/A
VI_ATTR_WIN_SIZE	N/A	N/A

2.8 Resource Summary Information

2.8.1 Summary of Attributes

This specification adds several attributes to the INSTR resource. No new attributes were defined in this specification for any other resource. The following is the full list of attributes defined for a VISA INSTR resource:

INSTR Resource

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_XON_CHAR
VI_ATTR_ASRL_XOFF_CHAR	VI_ATTR_GPIB_REN_STATE
VI_ATTR_CMDR_LA	VI_ATTR_DEST_ACCESS_PRIV
VI_ATTR_DEST_BYTE_ORDER	VI_ATTR_DEST_INCREMENT
VI_ATTR_FDC_CHNL	VI_ATTR_FDC_GEN_SIGNAL_EN
VI_ATTR_FDC_MODE	VI_ATTR_FDC_USE_PAIR
VI_ATTR_GPIB_PRIMARY_ADDR	VI_ATTR_GPIB_READDR_EN
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_MEM_BASE	VI_ATTR_MEM_SIZE
VI_ATTR_MEM_SPACE	VI_ATTR_MODEL_CODE
VI_ATTR_PXI_BUS_NUM	VI_ATTR_PXI_CHASSIS
VI_ATTR_PXI_DEV_NUM	VI_ATTR_PXI_FUNC_NUM
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
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
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
VI_ATTR_PXI_SLOT_LBUS_LEFT	VI_ATTR_PXI_SLOT_LBUS_RIGHT
VI_ATTR_PXI_SLOTPATH	VI_ATTR_PXI_STAR_TRIG_BUS
VI_ATTR_PXI_STAR_TRIG_LINE	VI_ATTR_PXI_TRIG_BUS
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_TERMCHAR
VI_ATTR_TERMCHAR_EN	VI_ATTR_TMO_VALUE
VI_ATTR_TRIG_ID	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	VI_ATTR_WR_BUF_OPER_MODE
VI_ATTR_DMA_ALLOW_EN	VI_ATTR_VXI_TRIG_SUPPORT
VI_ATTR_VXI_DEV_CLASS	VI_ATTR_TCPIP_ADDR
VI_ATTR_MANF_NAME	VI_ATTR_TCPIP_HOSTNAME
VI_ATTR_FILE_APPEND_EN	VI_ATTR_TCPIP_PORT
VI_ATTR_MODEL_NAME	VI_ATTR_4882_COMPLIANT

```
VI_ATTR_USB_SERIAL_NUM          VI_ATTR_USB_INTFC_NUM
VI_ATTR_USB_MAX_INTR_SIZE       VI_ATTR_USB_PROTOCOL
VI_ATTR_RD_BUF_SIZE             VI_ATTR_WR_BUF_SIZE
```

2.8.2 Summary of Events

This specification adds one event to the INSTR resource. No new events were defined in this specification for any other resource. The following is the full list of events defined for a VISA INSTR resource:

INSTR Resource

```
VI_EVENT_IO_COMPLETION
VI_EVENT_PXI_INTR
VI_EVENT_SERVICE_REQ
VI_EVENT_TRIG
VI_EVENT_VXI_SIGP
VI_EVENT_VXI_VME_INTR
VI_EVENT_USB_INTR
```

2.8.3 Summary of Operations

No new operations were defined in this specification. The summary of operations in *VPP-4.3* is unaffected by this specification.

3. PXI Extensions to the VISA Implementation Specification for Textual Languages

This section defines the extensions to *VXIplug&play Systems Alliance* specification *VPP-4.3.2: VISA Implementation Specification for Textual Languages* that are required to support PXI systems. Subsection 3.2 of this specification corresponds to section 3 of the *VPP-4.3.2* Specification, adding language binding information for PXI. Subsection 3.3 of this specification corresponds to Appendix A of the *VPP-4.3.2* specification, adding additional PXI-specific definitions to the specified implementation files.

3.1 Introduction

The specification *VPP-4.3.2: VISA Implementation Specification for Textual Languages* defines the symbols and constants for a VISA implementation for ANSI C or Visual BASIC. To provide full compatibility compilation time and run time, every implementation must use these exact bindings. This section provides additional constants that a VISA implementation that supports PXI must use. Similar constants shall be used with ATEasy. These constants are declared in the `visa.drv` ATEasy driver that contains the VISA library definitions including functions, constants, and types.

3.2 Additional VISA Textual Language Bindings for PXI

3.2.1 Type Assignments, Operation Prototypes, Completion Codes and Error Codes

There are no additional types, operations, completion codes, or error codes defined for PXI functionality in a VISA implementation.

3.2.2 Attribute Values

Table 3-1 shows the additional attribute values for supporting PXI in all framework bindings.

Table 3-1. Attribute Values for Supporting PXI

Attribute Names	Values
VI_ATTR_PXI_DEV_NUM	3FFF0201h
VI_ATTR_PXI_FUNC_NUM	3FFF0202h
VI_ATTR_PXI_BUS_NUM	3FFF0205h
VI_ATTR_PXI_CHASSIS	3FFF0206h
VI_ATTR_PXI_SLOTPATH	BFFF0207h
VI_ATTR_PXI_SLOT_LBUS_LEFT	3FFF0208h
VI_ATTR_PXI_SLOT_LBUS_RIGHT	3FFF0209h
VI_ATTR_PXI_TRIG_BUS	3FFF020Ah
VI_ATTR_PXI_STAR_TRIG_BUS	3FFF020Bh
VI_ATTR_PXI_STAR_TRIG_LINE	3FFF020Ch
VI_ATTR_PXI_MEM_TYPE_BAR0	3FFF0211h
VI_ATTR_PXI_MEM_TYPE_BAR1	3FFF0212h
VI_ATTR_PXI_MEM_TYPE_BAR2	3FFF0213h

Table 3-1. Attribute Values for Supporting PXI (Continued)

Attribute Names	Values
VI_ATTR_PXI_MEM_TYPE_BAR3	3FFF0214h
VI_ATTR_PXI_MEM_TYPE_BAR4	3FFF0215h
VI_ATTR_PXI_MEM_TYPE_BAR5	3FFF0216h
VI_ATTR_PXI_MEM_BASE_BAR0	3FFF0221h
VI_ATTR_PXI_MEM_BASE_BAR1	3FFF0222h
VI_ATTR_PXI_MEM_BASE_BAR2	3FFF0223h
VI_ATTR_PXI_MEM_BASE_BAR3	3FFF0224h
VI_ATTR_PXI_MEM_BASE_BAR4	3FFF0225h
VI_ATTR_PXI_MEM_BASE_BAR5	3FFF0226h
VI_ATTR_PXI_MEM_SIZE_BAR0	3FFF0231h
VI_ATTR_PXI_MEM_SIZE_BAR1	3FFF0232h
VI_ATTR_PXI_MEM_SIZE_BAR2	3FFF0233h
VI_ATTR_PXI_MEM_SIZE_BAR3	3FFF0234h
VI_ATTR_PXI_MEM_SIZE_BAR4	3FFF0235h
VI_ATTR_PXI_MEM_SIZE_BAR5	3FFF0236h

RULE: All attribute codes specified in Table 3-1 SHALL appear in the `visa.h` and `visa32.bas` files, and in the `visa.drv` file for ATEasy.

RULE: The `visa.h`, `visa32.bas`, and `visa.drv` files SHALL define all the attribute codes to be the same bit pattern as those in Table 3-1.

3.2.3 Event Type Values

Table 3-2 shows the additional event type value for supporting PXI in all framework bindings.

Table 3-2. Event Type Values for Supporting PXI

Attribute Name	Value
VI_EVENT_PXI_INTR	3FFF2022h

RULE: All event types specified in Table 3-2 SHALL appear in the `visa.h` and `visa32.bas` files, and in the `visa.drv` file for ATEasy.

RULE: The `visa.h`, `visa32.bas`, and `visa.drv` files SHALL define the event type to be the same bit pattern as those in Table 3-2.

3.2.4 Values and Ranges

Table 3-3 shows the additional values used for supporting PXI in all framework bindings.

Table 3-3. Additional Values for Supporting PXI

Name	Value
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_PROT_RESERVE	6
VI_TRIG_PROT_UNRESERVE	7
VI_PXI_STAR_TRIG_LINE_UNKNOWN	-1
VI_PXI_STAR_TRIG_CONTROLLER	1413
VI_PXI_LBUS_UNKNOWN	-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_UNKNOWN_CHASSIS	-1

3.2.5 Conditional Inclusion of PXI Functionality

VPP-4.3.2 Rule 3.8.4 states that a Vendor shall provide a macro for conditionally enabling extensions to the VISA Library. The macro for PXI extensions is `PXISAVISA_PXI`.

RULE: A VISA implementation that supports PXI SHALL provide interface files that enable PXI support when the macro `PXISAVISA_PXI` is defined.

OBSERVATION: Conditional compilation does not apply to ATEasy, as ATEasy always includes the VISA for PXI support.

3.3 Additions to the Implementation files

VPP-4.3.2 provides implementations of several files for binding the VISA library to ANSI C, Visual BASIC, and ATEasy. This section provides additions that a VISA implementation that supports PXI must add to those files.

3.3.1 Additional Contents of the visa.h File

This addition to the `visa.h` header file reflects the required implementation of the specifications given in this section.

```
#if defined(PXISAVISA_PXI)

#define VI_ATTR_PXI_DEV_NUM         (0x3FFF0201UL)
#define VI_ATTR_PXI_FUNC_NUM        (0x3FFF0202UL)
#define VI_ATTR_PXI_BUS_NUM         (0x3FFF0205UL)
#define VI_ATTR_PXI_CHASSIS        (0x3FFF0206UL)
#define VI_ATTR_PXI_SLOTPATH       (0xBFFF0207UL)
#define VI_ATTR_PXI_SLOT_LBUS_LEFT  (0x3FFF0208UL)
#define VI_ATTR_PXI_SLOT_LBUS_RIGHT (0x3FFF0209UL)
#define VI_ATTR_PXI_TRIG_BUS        (0x3FFF020AUL)
#define VI_ATTR_PXI_STAR_TRIG_BUS   (0x3FFF020BUL)
#define VI_ATTR_PXI_STAR_TRIG_LINE  (0x3FFF020CUL)

#define VI_ATTR_PXI_MEM_TYPE_BAR0   (0x3FFF0211UL)
#define VI_ATTR_PXI_MEM_TYPE_BAR1   (0x3FFF0212UL)
#define VI_ATTR_PXI_MEM_TYPE_BAR2   (0x3FFF0213UL)
#define VI_ATTR_PXI_MEM_TYPE_BAR3   (0x3FFF0214UL)
#define VI_ATTR_PXI_MEM_TYPE_BAR4   (0x3FFF0215UL)
#define VI_ATTR_PXI_MEM_TYPE_BAR5   (0x3FFF0216UL)

#define VI_ATTR_PXI_MEM_BASE_BAR0   (0x3FFF0221UL)
#define VI_ATTR_PXI_MEM_BASE_BAR1   (0x3FFF0222UL)
#define VI_ATTR_PXI_MEM_BASE_BAR2   (0x3FFF0223UL)
#define VI_ATTR_PXI_MEM_BASE_BAR3   (0x3FFF0224UL)
#define VI_ATTR_PXI_MEM_BASE_BAR4   (0x3FFF0225UL)
#define VI_ATTR_PXI_MEM_BASE_BAR5   (0x3FFF0226UL)

#define VI_ATTR_PXI_MEM_SIZE_BAR0   (0x3FFF0231UL)
#define VI_ATTR_PXI_MEM_SIZE_BAR1   (0x3FFF0232UL)
#define VI_ATTR_PXI_MEM_SIZE_BAR2   (0x3FFF0233UL)
#define VI_ATTR_PXI_MEM_SIZE_BAR3   (0x3FFF0234UL)
#define VI_ATTR_PXI_MEM_SIZE_BAR4   (0x3FFF0235UL)
#define VI_ATTR_PXI_MEM_SIZE_BAR5   (0x3FFF0236UL)
```

```
#define VI_EVENT_PXI_INTR           (0x3FFF2022UL)

#define VI_INTF_PXI                 (5)

#define VI_PXI_ALLOC_SPACE          (9)
#define VI_PXI_CFG_SPACE            (10)
#define VI_PXI_BAR0_SPACE           (11)
#define VI_PXI_BAR1_SPACE           (12)
#define VI_PXI_BAR2_SPACE           (13)
#define VI_PXI_BAR3_SPACE           (14)
#define VI_PXI_BAR4_SPACE           (15)
#define VI_PXI_BAR5_SPACE           (16)

#define VI_PXI_ADDR_NONE            (0)
#define VI_PXI_ADDR_MEM             (1)
#define VI_PXI_ADDR_IO              (2)
#define VI_PXI_ADDR_CFG             (3)

#define VI_TRIG_PROT_RESERVE        (6)
#define VI_TRIG_PROT_UNRESERVE      (7)

/* Values for VI_ATTR_PXI_STAR_TRIG_LINE */
#define VI_PXI_STAR_TRIG_LINE_UNKNOWN  (-1)
#define VI_PXI_STAR_TRIG_CONTROLLER    (1413)

/* Values for VI_ATTR_PXI_SLOT_LBUS_LEFT and VI_ATTR_PXI_SLOT_LBUS_RIGHT */
#define VI_PXI_LBUS_UNKNOWN            (-1)
#define VI_PXI_LBUS_STAR_TRIG_BUS_0    (1000)
#define VI_PXI_LBUS_STAR_TRIG_BUS_1    (1001)
#define VI_PXI_LBUS_STAR_TRIG_BUS_2    (1002)
#define VI_PXI_LBUS_STAR_TRIG_BUS_3    (1003)
#define VI_PXI_LBUS_STAR_TRIG_BUS_4    (1004)
#define VI_PXI_LBUS_STAR_TRIG_BUS_5    (1005)
#define VI_PXI_LBUS_STAR_TRIG_BUS_6    (1006)
#define VI_PXI_LBUS_STAR_TRIG_BUS_7    (1007)
#define VI_PXI_LBUS_STAR_TRIG_BUS_8    (1008)
#define VI_PXI_LBUS_STAR_TRIG_BUS_9    (1009)

#define VI_UNKNOWN_CHASSIS             (-1)

#endif
```

3.3.2 Additional Contents of the visa32.bas File

```
Global Const VI_ATTR_PXI_DEV_NUM         = &H3FFF0201&
Global Const VI_ATTR_PXI_FUNC_NUM        = &H3FFF0202&
Global Const VI_ATTR_PXI_BUS_NUM         = &H3FFF0205&
Global Const VI_ATTR_PXI_CHASSIS        = &H3FFF0206&
Global Const VI_ATTR_PXI_SLOTPATH       = &HBFFF0207&
Global Const VI_ATTR_PXI_SLOT_LBUS_LEFT  = &H3FFF0208&
Global Const VI_ATTR_PXI_SLOT_LBUS_RIGHT = &H3FFF0209&
Global Const VI_ATTR_PXI_TRIG_BUS        = &H3FFF020A&
Global Const VI_ATTR_PXI_STAR_TRIG_BUS   = &H3FFF020B&
```

```
Global Const VI_ATTR_PXI_STAR_TRIG_LINE  = &H3FFF020C&

Global Const VI_ATTR_PXI_MEM_TYPE_BAR0   = &H3FFF0211&
Global Const VI_ATTR_PXI_MEM_TYPE_BAR1   = &H3FFF0212&
Global Const VI_ATTR_PXI_MEM_TYPE_BAR2   = &H3FFF0213&
Global Const VI_ATTR_PXI_MEM_TYPE_BAR3   = &H3FFF0214&
Global Const VI_ATTR_PXI_MEM_TYPE_BAR4   = &H3FFF0215&
Global Const VI_ATTR_PXI_MEM_TYPE_BAR5   = &H3FFF0216&

Global Const VI_ATTR_PXI_MEM_BASE_BAR0   = &H3FFF0221&
Global Const VI_ATTR_PXI_MEM_BASE_BAR1   = &H3FFF0222&
Global Const VI_ATTR_PXI_MEM_BASE_BAR2   = &H3FFF0223&
Global Const VI_ATTR_PXI_MEM_BASE_BAR3   = &H3FFF0224&
Global Const VI_ATTR_PXI_MEM_BASE_BAR4   = &H3FFF0225&
Global Const VI_ATTR_PXI_MEM_BASE_BAR5   = &H3FFF0226&

Global Const VI_ATTR_PXI_MEM_SIZE_BAR0   = &H3FFF0231&
Global Const VI_ATTR_PXI_MEM_SIZE_BAR1   = &H3FFF0232&
Global Const VI_ATTR_PXI_MEM_SIZE_BAR2   = &H3FFF0233&
Global Const VI_ATTR_PXI_MEM_SIZE_BAR3   = &H3FFF0234&
Global Const VI_ATTR_PXI_MEM_SIZE_BAR4   = &H3FFF0235&
Global Const VI_ATTR_PXI_MEM_SIZE_BAR5   = &H3FFF0236&

Global Const VI_EVENT_PXI_INTR           = &H3FFF2022&

Global Const VI_INTF_PXI                 = 5

Global Const VI_PXI_ALLOC_SPACE          = 9
Global Const VI_PXI_CFG_SPACE            = 10
Global Const VI_PXI_BAR0_SPACE           = 11
Global Const VI_PXI_BAR1_SPACE           = 12
Global Const VI_PXI_BAR2_SPACE           = 13
Global Const VI_PXI_BAR3_SPACE           = 14
Global Const VI_PXI_BAR4_SPACE           = 15
Global Const VI_PXI_BAR5_SPACE           = 16

Global Const VI_PXI_ADDR_NONE            = 0
Global Const VI_PXI_ADDR_MEM             = 1
Global Const VI_PXI_ADDR_IO              = 2
Global Const VI_PXI_ADDR_CFG             = 3

Global Const VI_TRIG_PROT_RESERVE        = 6
Global Const VI_TRIG_PROT_UNRESERVE      = 7

' - Values for VI_ATTR_PXI_STAR_TRIG_LINE
Global Const VI_PXI_STAR_TRIG_LINE_UNKNOWN  = -1
Global Const VI_PXI_STAR_TRIG_CONTROLLER    = 1413

' - Values for VI_ATTR_PXI_SLOT_LBUS_LEFT and VI_ATTR_PXI_SLOT_LBUS_RIGHT
Global Const VI_PXI_LBUS_UNKNOWN            = -1
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_0    = 1000
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_1    = 1001
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_2    = 1002
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_3    = 1003
```

```
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_4    = 1004
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_5    = 1005
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_6    = 1006
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_7    = 1007
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_8    = 1008
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_9    = 1009

Global Const VI_UNKNOWN_CHASSIS             = -1
```

3.3.3 Additional Contents of the ATEasy visa.drv File

```
Global Const VI_ATTR_PXI_DEV_NUM = 0x3FFF0201
Global Const VI_ATTR_PXI_FUNC_NUM = 0x3FFF0202
Global Const VI_ATTR_PXI_BUS_NUM = 0x 3FFF0205
Global Const VI_ATTR_PXI_CHASSIS = 0x3FFF0206
Global Const VI_ATTR_PXI_SLOTPATH = 0xBFFF0207
Global Const VI_ATTR_PXI_SLOT_LBUS_LEFT = 0x3FFF0208
Global Const VI_ATTR_PXI_SLOT_LBUS_RIGHT = 0x3FFF0209
Global Const VI_ATTR_PXI_TRIG_BUS = 0x3FFF020A
Global Const VI_ATTR_PXI_STAR_TRIG_BUS = 0x3FFF020B
Global Const VI_ATTR_PXI_STAR_TRIG_LINE = 0x3FFF020C
Global Const VI_ATTR_PXI_MEM_TYPE_BAR0 = 0x3FFF0211
Global Const VI_ATTR_PXI_MEM_TYPE_BAR1 = 0x3FFF0212
Global Const VI_ATTR_PXI_MEM_TYPE_BAR2 = 0x3FFF0213
Global Const VI_ATTR_PXI_MEM_TYPE_BAR3 = 0x3FFF0214
Global Const VI_ATTR_PXI_MEM_TYPE_BAR4 = 0x3FFF0215
Global Const VI_ATTR_PXI_MEM_TYPE_BAR5 = 0x3FFF0216
Global Const VI_ATTR_PXI_MEM_BASE_BAR0 = 0x3FFF0221
Global Const VI_ATTR_PXI_MEM_BASE_BAR1 = 0x3FFF0222
Global Const VI_ATTR_PXI_MEM_BASE_BAR2 = 0x3FFF0223
Global Const VI_ATTR_PXI_MEM_BASE_BAR3 = 0x3FFF0224
Global Const VI_ATTR_PXI_MEM_BASE_BAR4 = 0x3FFF0225
Global Const VI_ATTR_PXI_MEM_BASE_BAR5 = 0x3FFF0226
Global Const VI_ATTR_PXI_MEM_SIZE_BAR0 = 0x3FFF0231
Global Const VI_ATTR_PXI_MEM_SIZE_BAR1 = 0x3FFF0232
Global Const VI_ATTR_PXI_MEM_SIZE_BAR2 = 0x3FFF0233
Global Const VI_ATTR_PXI_MEM_SIZE_BAR3 = 0x3FFF0234
Global Const VI_ATTR_PXI_MEM_SIZE_BAR4 = 0x3FFF0235
Global Const VI_ATTR_PXI_MEM_SIZE_BAR5 = 0x3FFF0236
Global Const VI_EVENT_PXI_INTR = 0x3FFF2022
Global Const VI_INTF_PXI = 5
Global Const VI_PXI_ALLOC_SPACE = 9
Global Const VI_PXI_CFG_SPACE = 10
Global Const VI_PXI_BAR0_SPACE = 11
Global Const VI_PXI_BAR1_SPACE = 12
Global Const VI_PXI_BAR2_SPACE = 13
Global Const VI_PXI_BAR3_SPACE = 14
Global Const VI_PXI_BAR4_SPACE = 15
Global Const VI_PXI_BAR5_SPACE = 16
Global Const VI_PXI_ADDR_NONE = 0
Global Const VI_PXI_ADDR_MEM = 1
Global Const VI_PXI_ADDR_IO = 2
Global Const VI_PXI_ADDR_CFG = 3
Global Const VI_TRIG_PROT_RESERVE = 6
```

```
Global Const VI_TRIG_PROT_UNRESERVE = 7

Global Const VI_PXI_STAR_TRIG_LINE_UNKNOWN = -1
Global Const VI_PXI_STAR_TRIG_CONTROLLER = 1413

Global Const VI_PXI_LBUS_UNKNOWN = -1
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_0 = 1000
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_1 = 1001
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_2 = 1002
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_3 = 1003
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_4 = 1004
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_5 = 1005
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_6 = 1006
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_7 = 1007
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_8 = 1008
Global Const VI_PXI_LBUS_STAR_TRIG_BUS_9 = 1009
Global Const VI_UNKNOWN_CHASSIS = -1
```

4. PXI Extensions to the VISA Implementation Specification for the G Language

This section defines the extensions to *VXIplug&play Systems Alliance* specification *VPP-4.3.3: VISA Implementation Specification for the G Language* that are required to support PXI systems. Subsection 4.2 of this specification corresponds to section 3 of the *VPP-4.3.3 Specification*, adding language-binding information for PXI.

4.1 Introduction

The specification *VPP-4.3.3: VISA Implementation Specification for the G Language* defines the symbols and constants for a VISA implementation for the G Language. To provide full compatibility compilation time and run time, every implementation must use these exact bindings. This section provides additional constants that must be used by a VISA implementation that supports PXI.

4.2 Additional VISA G Language Bindings for PXI

4.2.1 Type Assignments, Operation Prototypes, Completion Codes and Error Codes

There are no additional types, operations, completion codes, or error codes defined for PXI functionality in a VISA implementation.

4.2.2 Attribute Values

Table 4-1 lists the G Language binding and access privilege for each additional VISA attribute for a VISA Implementation that supports PXI.

Table 4-1. G Language Binding and Access Privileges

Attribute Names	G Language Binding	Access Privilege
VI_ATTR_PXI_DEV_NUM	PXI Device Number	Read Only
VI_ATTR_PXI_FUNC_NUM	PXI Function Number	Read Only
VI_ATTR_PXI_BUS_NUM	PXI 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	BAR0 Address Type	Read Only
VI_ATTR_PXI_MEM_TYPE_BAR1	BAR1 Address Type	Read Only

Table 4-1. G Language Binding and Access Privileges (Continued)

Attribute Names	G Language Binding	Access Privilege
VI_ATTR_PXI_MEM_TYPE_BAR2	BAR2 Address Type	Read Only
VI_ATTR_PXI_MEM_TYPE_BAR3	BAR3 Address Type	Read Only
VI_ATTR_PXI_MEM_TYPE_BAR4	BAR4 Address Type	Read Only
VI_ATTR_PXI_MEM_TYPE_BAR5	BAR5 Address Type	Read Only
VI_ATTR_PXI_MEM_BASE_BAR0	BAR0 Address Base	Read Only
VI_ATTR_PXI_MEM_BASE_BAR1	BAR1 Address Base	Read Only
VI_ATTR_PXI_MEM_BASE_BAR2	BAR2 Address Base	Read Only
VI_ATTR_PXI_MEM_BASE_BAR3	BAR3 Address Base	Read Only
VI_ATTR_PXI_MEM_BASE_BAR4	BAR4 Address Base	Read Only
VI_ATTR_PXI_MEM_BASE_BAR5	BAR5 Address Base	Read Only
VI_ATTR_PXI_MEM_SIZE_BAR0	BAR0 Address Size	Read Only
VI_ATTR_PXI_MEM_SIZE_BAR1	BAR1 Address Size	Read Only
VI_ATTR_PXI_MEM_SIZE_BAR2	BAR2 Address Size	Read Only
VI_ATTR_PXI_MEM_SIZE_BAR3	BAR3 Address Size	Read Only
VI_ATTR_PXI_MEM_SIZE_BAR4	BAR4 Address Size	Read Only
VI_ATTR_PXI_MEM_SIZE_BAR5	BAR5 Address Size	Read Only

4.2.3 Event Type Values

Table 4-2 shows the additional event type value for supporting PXI in all G-based framework bindings.

Table 4-2. Event Type Values for Supporting PXI

Attribute Name	Value
VI_EVENT_PXI_INTR	0x3FFF2022h

4.2.4 Values and Ranges

Table 4-3 shows the additional values for supporting PXI in all G-based framework bindings.

Table 4-3. Additional Values for Supporting PXI

Name	Value
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

Table 4-3. Additional Values for Supporting PXI (Continued)

Name	Value
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_PROT_RESERVE	6
VI_TRIG_PROT_UNRESERVE	7
VI_PXI_STAR_TRIG_LINE_UNKNOWN	-1
VI_PXI_STAR_TRIG_CONTROLLER	1413
VI_PXI_LBUS_UNKNOWN	-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_UNKNOWN_CHASSIS	-1