Systems Alliance

VPP-4.3.4: VISA Implementation Specification for COM

October 16, 2013

Revision 5.2

Systems Alliance

VPP-4.3.4 Revision History

This section is an overview of the revision history of the VPP-4.3.4 specification.

Revision 0.1, July 17, 2000
Original VPP-4.3.4 The COM I/O Libraries document. Based on VISA 2.2 specification

Revision 0.2 August 2, 2000
Revised specification based on notes from July 24-25 VXIpnp TWG meeting.

Revision 0.3 October 2, 2000
Revised specification based on notes from August VXIpnp TWG meeting.

Revision 0.4 November 15, 2000
Cleaned up various formatting issues.

Revision 0.5 January 5, 2001
Added feedback from November VXIpnp TWG meeting.

Revision 1.0 Draft February 5, 2001
Added feedback from posting of January 5 document to the IVI list server. Prepared for vote.

Revision 1.0 Draft May 1, 2001
Revised specification based on notes from February VXIpnp TWG meeting.

Revision 1.0 December 13, 2001
Removed the word “Draft” from the document and modified the date to show when it was final.

Revision 3.0 Draft, January 28, 2003
Added USB resource type and updated VXI, Resource Manager to achieve VISA 3.0 compliance.

Revision 3.0, January 15, 2004
Approved at IVI Board of Directors meeting.

Revision 3.1 Draft, May 10, 2005
Fixed tables in section 6 to be consistent with IDL. Changed GUID of custom marshaller category to be unique.

Revision 3.1 Draft, May 16, 2006
Added installation information regarding shared installer.

Revision 3.1 Draft, October 25, 2006

Adds Windows Vista to the list of supported operating systems. Fixes the split IDL problem from earlier draft.

Revision 3.2, February 14, 2008

Updated the introduction to reflect the IVI Foundation organization changes. Replaced Notice with text used by IVI Foundation specifications. Changed default installation directory to <Program Files>\IVI Foundation\VISA. Added comment to explain the intent of specific formatted I/O behavior.

Revision 4.0, October 16, 2008

Removed the description of the VISA COM Standard Components installer, which will be replaced by the VISA Shared Components installer described in VPP-4.3.5.

Revision 5.0, June 9, 2010

Added 64-bit integer support. Added HiSLIP features. Added PXI interface. Added Windows 7 to the list of supported operating systems.

Revision 5.1, March 6, 2013

Added Windows 8 to the list of supported operating systems.

Revision 5.2, October 16, 2013

Add interface IRegister64_2 to correct the signature of two methods.

NOTICE

VPP-4.3.4: *VISA Implementation Specification for COM* 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

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 COM I/O Library Specification

This section introduces the VISA specification. The VISA 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 COM I/O 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 this specification
- Audience for this 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

The VISA COM I/O specification provides a common standard for the IVI Foundation for developing multi-vendor software programs, including instrument drivers. This specification describes the VISA COM I/O architectural model, the configuration model, the interface definition language (IDL) file contents, and their semantics, which will usually be an annotated link to the VPP4-3 document, the VISA Library Specification.

VISA COM I/O, like the VISA library, 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. IVI COM drivers written to the VISA COM I/O specifications can execute within the IVI framework on systems that have the IVI COM libraries.

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 wishes to implement instrument driver software that is compliant with the VXI*plug&play* standards. The second audience is I/O vendors who wish to implement VISA-compliant I/O software. The third audience is instrumentation end users and application programmers who wish 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, *Introduction to the VXIplug&play Systems Alliance and the IVI Foundation*, explains the VXI*plug&play* Systems Alliance and its relation to the IVI Foundation.

Section 2, *Overview of VISA COM I/O Library Specification*, provides an overview of this specification, including the objectives, scope and organization, application, references, definition of terms and acronyms, and conventions.

Section 3, *VISA Resource Template: IVisaSession and IEventManager*, describes COM interfaces implementing the VISA Resource Template.

Section 4, *VISA COM I/O Resource Management*, describes the COM interfaces and components that comprise the VISA COM I/O Resource Manager as well as the Init() method of the IVisaResource interface.

Section 5, *VISA COM I/O Resource Classes*, presents the COM interfaces for specific instrument resources.

Section 6, *VISA COM I/O Components and Installation*, discusses implementation of VISA COM I/O Components.

Section 7, *Formatted I/O*, presents the Formatted I/O interface(s) for VISA COM I/O.

Section 8, *The Complete VISA COM I/O IDL*, presents the complete IDL specification for the VISA COM I/O Libraries.

2.4. Application of This Specification

This specification is intended for use by developers of IVI COM instrument drivers and by developers of VISA COM I/O Libraries software. It is also useful as a reference for end users of IVI COM instrument drivers. This specification is intended for use in conjunction with the IVI Instrument Driver Specifications including the architecture and technology specifications (IVI-3.*x*) and the instrument class driver specifications (IVI-4.*x*). 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 the aforementioned IVI specifications and VXI plug&play VPP-3.*x* specifications can be used in a wide variety of higher-level software environments, as described in the *System Frameworks Specification* (VPP-2).

2.5. Microsoft COM and the VISA API

The VISA COM I/O API has a few basic rules that apply across all the interfaces and components in order to be COM compliant.

RULE 2.5.1
All VISA COM I/O Interfaces and Components **SHALL** be COM-compliant.

RULE 2.5.2
All VISA COM I/O Components **SHALL** operate in both STA and MTA apartments **AND SHALL** be registered as "Both" in the system registry.

PERMISSION 2.5.1
VISA COM I/O Components **MAY** use the free-threaded marshaller.

OBSERVATION 2.5.1
STA stands for Single Threaded Apartment and MTA stands for Multi Threaded Apartment.

OBSERVATION 2.5.2
See Section 6 for additional rules and recommendations for marshalling techniques.

2.6. VISA COM I/O Implementation and Distribution Requirements

VISA COM I/O Implementations will redistribute several shared global files and will also provide some vendor-specific components. The very minimum compliant installation would provide the VXI*plug&play*-owned Global Resource Manager (GRM) and Formatted I/O components and their associated files and a Vendor-Specific Resource Manager (SRM) with one VISA COM I/O Resource Component that implements IVisaSession and IEventManager.

Example 1:

If a vendor wanted to provide a driver for a PC plug-in card that allowed SCPI string communication, it would redistribute the global shared components, provide an SRM that knows how to instantiate the plug-in's resource, and provide a VISA COM I/O resource for the plug-in that implements IMessage, IAsyncMessage, IVisaSession, and IEventManager COM interfaces.

Example 2:

If a vendor wished to provide a VISA COM I/O implementation that could create ASRL INSTR and GPIB INSTR sessions, they would redistribute the global shared components and provide an SRM that can parse both kinds of address strings and can find and create resources of both types. They would also provide two different VISA COM I/O Resource Components, one that implemented ISerial, the IMessage interfaces, and the two base interfaces and another that implemented IGpib, the IMessage interfaces, and the two base interfaces.

Table 2.6.1 shows a list of shared global files to be redistributed.

Component Name	Description
`Global Resource Manager (GRM)`	A DLL containing the Global Resource Manager COM Component and the VISA COM I/O shared type library resource.
`Basic Formatted I/O Component`	A DLL containing a component that implements the IFormattedIO488 interface.
`Conflict Table Manager Component (and Conflict Table)`	A DLL containing a component that implements the IVisaConflictTableManager interface and is used by the Global Resource Manager to resolve conflicts where multiple vendor components try to control a hardware resource.

Table 2.6.1

The installation rules and requirements for the Global Shared Components are listed in Section 6.1.

In addition to the shared global files, a VISA COM I/O implementation must provide several vendor-specific files to be compatible with the VISA COM I/O standard.

Table 2.6.2 shows a list of the required files and some optional files.

Component Name	Description
`Vendor-Specific Resource Manager (SRM)`	A DLL containing a resource manager COM Component that can find and instantiate all of the resources implemented by the vendor's VISA COM I/O implementation.
`One or more Resource Components`	One or more DLLs containing one or more COM Components that implement at least the IVisaSession and IEventManager interfaces.
`Vendor Help File (optional)`	A help file containing entries describing the errors returned by the Vendor's resources, information about the resources themselves, descriptions of any vendor-defined COM interfaces, and any additional information deemed appropriate by the vendor.
`Vendor Type Library (optional)`	A COM type library describing all the co-classes and COM interfaces and types defined by the vendor.

Table 2.6.2

The installation rules and requirements for the Vendor Specific Components are listed in Section 6.1.

OBSERVATION 2.6.1

Unlike VPP-4.3.2 and VPP-4.3.3, which rely on a single file named visa32.dll, a VISA COM I/O implementation has no name requirements. This allows both COM-based and non-COM-based implementations to reside side-by-side on the same system.

The following table shows the correspondence between the VISA data types specified in VPP-4.3 and the COM data types used in the IDL syntax in this specification. It is the intent of this specification that these data types be semantically equivalent where possible. Note that in some cases in this specification, enumerations are used rather than the generic '`short`' or '`long`' type. This improves both coding and readability. The size of each property's data type in bits is consistent with VPP-4.3. In this specification, the specifier '`v1_enum`' is used for 32-bit enumerations. This specification reserves the right to add additional values to existing enumerations without creating a new COM interface.

VISA Data Type	COM Data Type
ViUInt32	long
ViAUInt32	SAFEARRAY(long)
ViInt32	long
ViAInt32	SAFEARRAY(long)
ViUInt16	short
ViAUInt16	SAFEARRAY(short)
ViInt16	short
ViAInt16	SAFEARRAY(short)
ViUInt8	BYTE
ViAUInt8	SAFEARRAY(BYTE)
ViInt8	BYTE
ViAInt8	SAFEARRAY(BYTE)
ViByte	BYTE
ViAByte	SAFEARRAY(BYTE)
ViString	BSTR
ViAString	SAFEARRAY(BSTR)
ViRsrc	BSTR
ViARsrc	SAFEARRAY(BSTR)
ViChar	BYTE
ViBuf	SAFEARRAY(BYTE)

VISA Data Type	COM Data Type
ViVersion	long
ViAttr	long
ViConstString	BSTR
ViAccessMode	long
ViBusAddress	long
ViBusSize	long
ViAttrState	VARIANT
ViEventType	long
ViKeyId	BSTR
ViJobId	long
ViReal32	float
ViReal64	double
ViAddr	VARIANT
ViStatus	HRESULT
ViBoolean	VARIANT_BOOL
ViHndlr	IEventHandler
ViFindList	SAFEARRAY(BSTR)
ViSession	IVisaSession, IResourceManager, IResourceManager3
ViEvent	IEvent
ViObject	N/A

Table 2.6.3

2.7. 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*
- *NI-488.2 User Manual for DOS*, National Instruments Corporation
- *NI-488.2M User Manual*, National Instruments Corporation
- *NI-VXI Programmer Reference Manual*, National Instruments Corporation
- *NI-VXI User Manual*, National Instruments Corporation
- *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 Functional Body Specification*
- VPP-3.3, *Instrument Driver Interactive Developer Interface Specification*
- VPP-3.4, *Instrument Driver Programmatic Developer Interface Specification*
- VPP-4.1, *VISA-1 Main Specification*
- VPP-4.2, *The VISA Transition Library*
- VPP-4.3, *The VISA Library*
- VPP-4.3.2, *VISA Implementation Specification for Textual Languages*
- VPP-4.3.3, *VISA Implementation Specification for the G Language*
- VPP-6, *Installation and Packaging Specification*
- VPP-7, *Soft Front Panel Specification*
- VPP-9, *Instrument Vendor Abbreviations*
- VXI-1, *VXIbus System Specification*, Revision 1.4, VXIbus Consortium
- VXI-11, *TCP/IP Instrument Protocol*, VXIbus Consortium

2.8. 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.

API Application Programmers Interface. The direct interface that an end user sees when creating an application. The VISA API consists of the sum of all of the operations, attributes, and events of each of the VISA Resource Classes. The VISA COM I/O API consists of a collection of COM interfaces.

Attribute A value within a resource that reflects a characteristic of the operational state of a resource. Also known as a property.

COM **C**omponent **O**bject **M**odel, a Microsoft technology for reusable software components.

COM Class A software construct defined by Microsoft's COM specification that represents a logical object and has one or more interfaces, including `IUnknown`. See "Component" for more information about VISA COM I/O-compliant classes.

COM Interface A Microsoft COM term that refers to a specification of a group of related methods containing additional marshalling and other information that is similar to a class with no implementation in C++. COM Classes implement one or more interfaces, including the interface `IUnknown`.

COM Object A live instance of a COM Class.

Commander A device that has the ability to control another device. This term can also denote the unique device that has sole control over another device (as with the VXI Commander/Servant hierarchy).

Component A DLL or EXE that implements the COM entry points and can instantiate at least one COM Class. A VISA COM I/O Component is always a DLL and additionally requires that at least one instantiatable class implement the interface "`IVisaSession`" and whatever interfaces are appropriate or required for the resource type returned by `IVisaSession`'s `HardwareInterfaceType` property.

Communication Channel The same as *Session*. A communication path between a software element and a resource. Every communication channel in VISA COM I/O is unique. A Session in VISA COM I/O is an instance of a COM Class that implements IVisaSession and that has had `Init()` successfully called on it (either by a resource manager or directly) and has not yet had `Close()` called on it.

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 an INSTR resource.

Instrument A device that accepts some form of stimulus to perform a designated task, test, or measurement function. Two common forms of stimuli are message passing and register reads and writes. Other forms include triggering or varying forms of asynchronous control.

Instrument Driver	Library of functions for controlling a specific instrument
Interface	1. 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. 2. See "COM Interface".
MTA (Multi-Threaded Aparment)	A COM construct in which COM components live that permits multiple simultaneous method calls on the component's interfaces.
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.
Register	An address location that either contains a value that is a function of the state of hardware or can be written into to cause hardware to perform a particular action or to enter a particular state. In other words, an address location that controls and/or monitors hardware.
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. A Session in VISA COM I/O is an instance of a COM Class that implements IVisaSession and that has had `Init()` called on it and has not yet had `Close()` called on it.
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).
STA (Single-Threaded Aparment)	A COM construct in which COM components live that guarantees that the methods on a component's interfaces will be called serially, i.e., only one method call at a time.
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.
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.
VISA Resource Manager	This is the name given to the part of VISA that manages resources. This management includes support for opening, closing, and finding resources; setting attributes, retrieving attributes, and generating events on resources; and so on.
VISA Resource Template	This is the name given to the part of VISA defines the basic constraints and interface definition for the creation and use of a VISA resource. All VISA resources must derive their interface from the definition of the VISA Resource Template.

2.9. 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.

Recommendations consist of advice to implementers that will affect the usability of the final device. They are included in this standard to draw attention to particular characteristics that the authors believe to be important to end user success.

Permissions are included to *authorize* specific implementations or uses of system components. A permission is characterized by the use of the word **MAY** in bold upper case characters. These permissions are granted to ensure specific System Framework components are well defined and can be tested for compatibility and interoperability.

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 that 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 implementer 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 Resource Template: IVisaSession and IEventManager

VISA defines an architecture consisting of many resources that encapsulate device functionality. Each resource can give specialized services to applications or to other resources. Achieving this capability requires a high level of consistency in the operation of VISA resources. This level of consistency is achieved through a precisely defined, extensible interface, which provides a well-defined set of services. In VISA's C API, the resource template is a collection of methods and constant values. In VISA COM I/O the resource template is defined by the COM interface "IVisaSession." All VISA COM I/O resource COM interfaces derive from this base interface. This provides users a way to polymophically act on all resources and provides consistency across interfaces. The basic services available from the IVisaSession interface include the following:

- Creating and deleting sessions (Life Cycle Control)
- Modifying and retrieving individual resource characteristics called *Attributes* (Characteristic Control)
- Restricting resource access (Access Control)

3.1. Template Services

3.1.1. Control Services

The IVisaSession interface provides all the basic resource control services to applications. These basic services include controlling the life cycle of sessions to resources/devices and manipulating resource characteristics. A summary of these services is presented below:

- **Life Cycle Control**
  IVisaSession (along with the COM API) controls the life cycle of sessions, find lists, and events. A Session is defined as an instance of a COM component that implements IVisaSession. Once an application has finished using any of them, it can use the `Close()` method to free up all the system resources associated with it. Optionally, if the COM reference count for a resource component goes to zero, the resource will clean itself up. VISA COM I/O resources will free up all associated system resources when an application is abnormally terminated. Behavior of VISA COM I/O components in a process that is still active but damaged is undefined.
- **Characteristic Control**
  Resources can have attributes associated with them. Some attributes depict the instantaneous state of the resource and some define alterable parameters to modify the behavior of the resources. These attributes are defined by individual resources. VISA COM I/O provides access to all attributes through the methods `GetAttribute` and `SetAttribute`. For legacy reasons with VISA, attributes that are accessible from COM Property methods are also accessible by name through Get/Set Attribute.
- **Asynchronous Operation Control**
  Resources can have asynchronous operations associated with them. These operations are invoked in the same way that all other operations are invoked. Instead of waiting for the actual job to be done, they register the job to be done and return immediately. When the I/O is complete, an event is generated to indicate the completion status of the associated operation. Unlike VISA, there is no resource template-defined terminate operation. The only available `Terminate` is on the `IAsyncMessage` interface, described in Section 5.1.3, *IASyncMessage Interface*.
- **Access Control**
  Applications can open multiple sessions to a VISA COM I/O resource simultaneously. Applications can access the VISA COM I/O resource through the different sessions concurrently. However, in certain cases, an application accessing a VISA COM I/O resource might want to restrict other applications or sessions from accessing that resource. VISA defines a locking mechanism to restrict accesses to resources for such special circumstances. The operation used to acquire a lock on a resource is `LockRsrc()`, and the operation to relinquish the lock is `UnlockRsrc()`.

3.1.2. Communication Services

Applications using VISA COM I/O access resources by opening sessions to them. The primary method of communication to resources is by invoking methods on interfaces implemented by the session. A VISA COM I/O resource also allows information exchange through events.

- **Operation Invocation**
  After establishing a session, an application can communicate with it by invoking operations associated with the resources. All interfaces use COM error handling to report errors. There are some general HRESULT Error Codes below as well as codes specific to methods.

- **Event Reporting**
  VISA provides callback, queuing, and waiting services that can inform sessions about resource-defined events.

VISA COM Error Codes	VISA Error Codes	Description
E_VISA_INV_OBJECT	VI_ERROR_INV_SESSION VI_ERROR_INV_OBJECT	The given session or object reference is invalid.
E_VISA_NSUP_OPER	VI_ERROR_NSUP_OPER	The given session does not support this operation.
(not applicable)	VI_ERROR_NIMPL_OPER	The given operation is not implemented.
E_VISA_SYSTEM_ERROR	VI_ERROR_SYSTEM_ERROR	Unknown system error (miscellaneous error).
E_VISA_INV_PARAMETER	VI_ERROR_INV_PARAMETER	The value of some parameter—which parameter is not known—is invalid.
E_VISA_USER_BUF	VI_ERROR_USER_BUF	A specified user buffer is not valid or cannot be accessed for the required size.

Table 3.1.1

OBSERVATION 3.1.1

It is possible that in the future, any operation may return success or error codes not listed in this specification. Therefore, it is important that applications check for general success or failure before comparing a return value to known return codes.

OBSERVATION 3.1.2

It is the intention of this specification to have success and warning codes be greater than or equal to zero and error codes less than zero. The specific status values are specified in the corresponding framework documents. Only unique identifiers are specified in this document.

RECOMMENDATION 3.1.1

If an operation defines an error code for a given parameter, a VISA implementation should normally use that error code.

RULE 3.1.1

If a VISA COM I/O implementation cannot determine which parameter caused an error, such as when using a lower-level driver, then it **SHALL** return E_INVALIDARG or E_VISA_INV_PARAMETER.

RULE 3.1.2

If a VISA COM I/O resource is unable to allocate memory to satisfy a request, it **SHALL** return E_OUTOFMEMORY or E_VISA_ALLOC.

RULE 3.1.3

If a VISA COM I/O driver's internal data becomes corrupted or it encounters an internal logic error, it **SHALL** return E_UNEXPECTED or E_VISA_SYSTEM_ERROR.

RULE 3.1.4

If a VISA COM I/O resource receives an invalid pointer argument, it **SHALL** return E_POINTER or E_VISA_USER_BUF

RULE 3.1.5

A VISA COM I/O resource **SHALL NOT** return E_NOTIMPL or E_VISA_NIMPL_OPER.

RECOMMENDATION 3.1.2

In addition to a `HRESULT` error message upon an error, the resource should place an `IErrorInfo` structure on the thread with an error description (including the name of the bad parameter if that is the cause of the error) and a help file reference.

OBSERVATION 3.1.3

The above rules allow multiple status codes for the same error. It is possible for some of these errors to be caught by a COM proxy, where in other cases the lower-level driver may return the error. Either error code specified is compliant.

OBSERVATION 3.1.4

The VISA COM status codes listed above are semantically equivalent to the similarly named status codes in VPP 4.3.

RULE 3.1.6

A VISA COM I/O implementation **SHALL** convert a non-zero ViStatus value to an HRESULT status value by masking the value with 0x80000FFF and adding (bit-ORing) in the value 0x00040000.

OBSERVATION 3.1.5

The ViStatus value of 0 (VI_SUCCESS) is also an HRESULT value of 0 (S_OK or S_VISA_SUCCESS).

3.2. VISA Template Interface Overview

This section summarizes the interface that each VISA implementation must incorporate. The different attributes and operations are described in detail in subsequent sections.

3.2.1. VISA Template Attributes

RULE 3.2.1

VISA COM I/O resources **SHALL** follow the behaviors defined in section 3.2.1 of VPP 4.3 with the following exceptions.

RULE 3.2.2

The following attributes **SHALL NOT** be defined in VISA COM I/O: VI_ATTR_ RM_SESSION, VI_ATTR_USER_DATA.

OBSERVATION 3.2.1

The value of the attribute `VI_ATTR_RSRC_SPEC_VERSION` is a fixed value that reflects the version of the VISA COM I/O specification to which the resource implementation is compliant. This value will change with subsequent versions of the specification. This value will be identical or related to the VISA specification number that this specification is based on.

OBSERVATION 3.2.2

There may be resources with different specification versions residing on the same machine, and the resource manager may return components with different specification versions.

ViVersion Description for VI_ATTR_RSRC_IMPL_VERSION and VI_ATTR_RSRC_SPEC_VERSION

Bits 31 to 20	Bits 19 to 8	Bits 0 to 7
Major Number	Minor Number	Sub-minor Number

Table 3.2.1

3.2.2. IVisaSession Interface

The `IVisaSession` Interface is defined in IDL as follows.

```
   [
      object,
      oleautomation,
      helpstring("VISA Session Interface"),
      uuid(db8cbf03-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIVisaSession + 49),
      pointer_default(unique)
   ]
   interface IVisaSession : IUnknown
   {
      [propget, helpcontext(HlpCtxIVisaSession  + 1), helpstring("Get the
implementation version of the component")]
      HRESULT ComponentVersion([out, retval] long *pVal);
      [propget, helpcontext(HlpCtxIVisaSession  + 2), helpstring("Get the VISA
COM I/O specification version")]
      HRESULT SpecVersion([out, retval] long *pVal);
      [propget, helpcontext(HlpCtxIVisaSession  + 3), helpstring("Get a
description of the hardware interface")]
      HRESULT HardwareInterfaceName([out, retval] BSTR *pVal);
      [propget, helpcontext(HlpCtxIVisaSession  + 4), helpstring("Get the
hardware interface number")]
      HRESULT HardwareInterfaceNumber([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIVisaSession  + 5), helpstring("Get the
hardware interface type")]
      HRESULT HardwareInterfaceType([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIVisaSession  + 6), helpstring("Get the
```

```
current lock state of the resource")]
     HRESULT LockState([out, retval] AccessMode *pVal);
     [propget, helpcontext(HlpCtxIVisaSession  + 7), helpstring("Get the
current state of all settable properties")]
     HRESULT OptionString([out, retval] BSTR *pVal);
     [propget, helpcontext(HlpCtxIVisaSession  + 8), helpstring("Get the ProgID
of the component")]
     HRESULT ProgID([out, retval] BSTR *pVal);
     [propget, helpcontext(HlpCtxIVisaSession  + 9), helpstring("Get the
resource name")]
     HRESULT ResourceName([out, retval] BSTR *pVal);
     [propget, helpcontext(HlpCtxIVisaSession  + 10), helpstring("Get the
session class type")]
     HRESULT SessionType([out, retval] BSTR *pVal);
     [propget, helpcontext(HlpCtxIVisaSession  + 11), helpstring("Get the
manufacturer ID of the component")]
     HRESULT SoftwareManufacturerID([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIVisaSession  + 12), helpstring("Get the
manufacturer name of the component")]
     HRESULT SoftwareManufacturerName([out, retval] BSTR *pVal);
     [propget, helpcontext(HlpCtxIVisaSession  + 13), helpstring("Get/Set the
I/O timeout in milliseconds")]
     HRESULT Timeout([out, retval] long *pVal);
     [propput, helpcontext(HlpCtxIVisaSession  + 13), helpstring("Get/Set the
I/O timeout in milliseconds")]
     HRESULT Timeout([in] long newVal);
     [propget, helpcontext(HlpCtxIVisaSession  + 14), helpstring("Get the last
status from this session")]
     HRESULT LastStatus([out, retval] HRESULT *pVal);

     [hidden, helpcontext(HlpCtxIVisaSession  + 15), helpstring("Get the state
of a specified property")]
     HRESULT GetAttribute(
        [in] long attribute,
        [out, retval] VARIANTARG *pAttrState);
     [hidden, helpcontext(HlpCtxIVisaSession  + 16), helpstring("Set the state
of a specified property")]
     HRESULT SetAttribute(
        [in] long attribute,
        [in] VARIANTARG attrState);
     [helpcontext(HlpCtxIVisaSession  + 17), helpstring("Establish ownership of
the resource")]
     HRESULT LockRsrc(
        [in, defaultvalue(EXCLUSIVE_LOCK)] AccessMode type,
        [in, defaultvalue(2000)] long lockTimeout,
        [in, defaultvalue("")] BSTR requestedKey,
        [out, retval] BSTR *pAccessKey);
     [helpcontext(HlpCtxIVisaSession  + 18), helpstring("Relinquish ownership
of the resource")]
     HRESULT UnlockRsrc();
     [helpcontext(HlpCtxIVisaSession  + 19), helpstring("Initialize a session
to the specified resource name")]
     HRESULT Init(
        [in] BSTR resourceName,
        [in, defaultvalue(NO_LOCK)] AccessMode mode,
        [in, defaultvalue(2000)] long initTimeout,
        [in, defaultvalue("")] BSTR optionString);
     [helpcontext(HlpCtxIVisaSession  + 20), helpstring("Close the session")]
     HRESULT Close();
  };
```

The `IVisaSession` Interface has several COM properties that correspond to attributes defined in VISA. The following table shows property-attribute equivalence for `IVisaSession`.

Property Name	VISA Attribute Name

ComponentVersion	VI_ATTR_RSRC_IMPL_VERSION
SpecVersion	VI_ATTR_RSRC_SPEC_VERSION
LockState	VI_ATTR_RSRC_LOCK_STATE
SoftwareManufacturerID	VI_ATTR_RSRC_MANF_ID
SoftwareManufacturerName	VI_ATTR_RSRC_MANF_NAME
Timeout	VI_ATTR_TMO_VALUE
HardwareInterfaceNumber	VI_ATTR_INTF_NUM
HardwareInterfaceName	VI_ATTR_INTF_NAME
HardwareInterfaceType	VI_ATTR_INTF_TYPE
ResourceName	VI_ATTR_RSRC_NAME
SessionType	VI_ATTR_RSRC_CLASS
OptionString	No VISA equivalent
ProgID	No VISA equivalent
LastStatus	No VISA equivalent

Table 3.2.2

The IVisaSession Interface has several methods that map to VISA functions. The following table shows VISA equivalence for IVisaSession methods.

Method Name	VISA Equivalent
LockRsrc	viLock
UnlockRsrc	viUnlock
Close	viClose
Init	viOpen

Table 3.2.3

OBSERVATION 3.2.3

There is not a one-to-one mapping between IVisaSession and the VISA Resource Template. Because the properties HardwareInterfaceNumber, Timeout, HardwareInterfaceName, and HardwareInterfaceType are used by all resource types, they have been moved up to IVisaSession to maximize polymorphism.

RULE 3.2.3

Every VISA COM I/O resource **SHALL** implement IVisaSession.

RULE 3.2.4

The Close() method **SHALL** cause the resource to clean itself up, but **SHALL NOT** destroy the COM object.

RULE 3.2.5

The OptionString property **SHALL** return the names and values of all the settable COM properties of all the interfaces derived from IVisaSession that a resource supports. The string **SHALL** follow the grammar described in RULE 3.2.9 and RULE 3.2.10 with the following additional restrictions: there **SHALL** be one space character between all tokens, with no other whitespace between, before, or after tokens, the string **SHALL** use the appropriate enumeration value names for properties that have corresponding enumerations, and boolean properties **SHALL** use the strings “TRUE” and “FALSE” (without quotation marks.)

OBSERVATION 3.2.4

The `OptionString` property is not identical to the `OptionString` parameter to the `Open()` and `Init()` methods, although it is possible in some instances that they may be the same value. This property consists of all settable properties, not just those that the user has explicitly initialized.

OBSERVATION 3.2.5

COM rules dictate that objects are destroyed only when their reference count goes to zero, and this can be difficult for a developer to determine or cause to happen in some environments. This means that `Close` is still necessary to deterministically destroy a resource.

RULE 3.2.6

The SessionType Property **SHALL** return the following values for the Session types. This is consistent with VPP-4.3.

Session Type Name	Session Type String
`::INSTR`	`INSTR`
`::INTFC`	`INTFC`
`::MEMACC`	`MEMACC`
`::BACKPLANE`	`BACKPLANE`
`::SOCKET`	`SOCKET`

Table 3.2.4

RULE 3.2.7

Each method of each COM interface derived from IVisaSession **SHALL** update its LastStatus property on each invocation with the HRESULT status being returned from said method.

3.2.2.1. HRESULT **Init**([in] BSTR resourceName, [in, defaultvalue(0)] AccessMode mode, [in, defaultvalue(2000)] long initTimeout, [in, defaultvalue("")] BSTR optionString)

Purpose

Open and Initialize a VISA COM I/O Resource.

Parameter

Name	Direction	VISA Type	COM Type	Description
resourceName	In	ViRsrc	BSTR	String that represents a legal VISA resource string.
mode	In	ViAccessMode	AccessMode	Request for locking privilege, legal values are: NO_LOCK and EXCLUSIVE_LOCK. The default is NO_LOCK
initTimeout	In	ViInt32	long	Parameter that represents the time to wait to open a resource if a lock is attempted.
optionString	In	ViString	BSTR	String that represents a list of initialization parameters for the object.

Table 3.2.5

Return Values

This function may return the HRESULT-equivalent of any VISA error codes documented for viOpen in VPP 4.3.

Description

This method opens a VISA session and optionally initializes its parameters through a string of name-value pairs. It is equivalent to VISA's viOpen command with the additional power of initialization.

Related Items

See the IVisaSession interface.

Implementation Requirements

RULE 3.2.8

The OptionString **SHALL** be compliant to the following grammar (with strings and numbers as defined at the top and whitespace optional and legal between all tokens:

```
STRING:             "[a-zA-Z_][0-9a-zA-Z_]*"
;
NUMBER:             [0-9]+
              |     0x[0-9a-fA-F]+
;
optionstring:       valuepair
              |     valuepair ; optionstring
;
valuepair:          propname = initvalue
;
propname:           STRING
;
initvalue:          STRING
              |     NUMBER
              |     boolean
;
boolean:            TRUE
              |     FALSE
;
```

RULE 3.2.9

The `OptionString` parameter **SHALL** support all the settable COM properties (specifically, the COM properties that have the IDL attribute "propput") of all the COM interfaces derived from IVisaSession a resource supports. The `Init()` method **SHALL** accept the propname names as the string names of the COM properties as defined in the COM IDL specification and each initvalue **SHALL** conform to the following rules:

Property Value Type	Valid Values
Numeric	Either 1. A numeric value as defined above. Or 2. If there is an enumeration defined the the VISACOM type library for the attribute, the string name of the enumeration value.
String	The string enclosed in quotation marks
Boolean	Either 1. A numeric value as defined above, with non-zero true and zero defined as false 2. The strings TRUE or FALSE.

Table 3.2.6

OBSERVATION 3.2.6

There are currently no COM properties that have the "propput" IDL attribute on interfaces that derive from IVisaSession that take string values.

RULE 3.2.10

The `Init` method **SHALL** fail with an error of E_VISA_INV_SETUP when it is called on an object that has already been initialized.

RULE 3.2.11

The `Init` method **SHALL** fail with an error of E_VISA_INV_SETUP when it is called on a resource that has already had `Close` called on it.

RULE 3.2.12
The `Init` method **SHALL** fail with an error of E_INVALIDARG when OptionString is malformed.

RULE 3.2.13
The `Init` method **SHALL** fail with an error of E_INVALIDARG when an unsupported initvalue is given for a valid propname.

RULE 3.2.14
The `Init` method **SHALL NOT** fail when a propname or initvalue is passed with wrong capitalization **UNLESS** propname/initvalue uniqueness is compromised or otherwise noted.

RULE 3.2.15
The `Init` method **SHALL** fail with an error of E_VISA_NSUP_ATTR due to an unsupported propname.

RULE 3.2.16
The `Init` method **SHALL** behave like a `viOpen` that only works for the VISA resource strings supported by the VISA COM I/O resource that implements the `Init`. Additionally, the `Init` string will cause properties of the resource to be set after a successful connection is established to the hardware resource.

RULE 3.2.17
The `Init` method **SHALL** fail with an error of E_VISA_ATTR_READONLY when the init string contains a read-only propname.

PERMISSION 3.2.1
A vendor-specific `Init` implementation **MAY** support the VISA-defined attribute names and values in the OptionString.

3.3. Event Services

3.3.1. IEventManager Interface

The `IEventManager` interface implements the portion of the VISA resource template dealing with asynchronous events. It, together with the `IVisaSession` interface, correspond to the VISA resource template in VISA COM I/O. The IEventManager interface is defined as follows:

```
   [
      object,
      oleautomation,
      helpstring("Event Manager Interface"),
      uuid(db8cbf14-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIEventManager + 49),
      pointer_default(unique)
   ]
   interface IEventManager : IVisaSession
   {
      [propget, helpcontext(HlpCtxIEventManager  + 1), helpstring("Get/Set the
queue length")]
      HRESULT MaximumQueueLength([out, retval] long *pVal);
      [propput, helpcontext(HlpCtxIEventManager  + 1), helpstring("Get/Set the
queue length")]
      HRESULT MaximumQueueLength([in] long newVal);

      [helpcontext(HlpCtxIEventManager  + 2), helpstring("Enable the specified
event")]
      HRESULT EnableEvent(
         [in] EventType type,
         [in] EventMechanism mech,
         [in, defaultvalue(0)] long customEventType);
      [helpcontext(HlpCtxIEventManager  + 3), helpstring("Disable the specified
event")]
      HRESULT DisableEvent(
         [in, defaultvalue(ALL_ENABLED_EVENTS)] EventType type,
         [in, defaultvalue(EVENT_ALL_MECH)] EventMechanism mech,
         [in, defaultvalue(0)] long customEventType);
      [helpcontext(HlpCtxIEventManager  + 4), helpstring("Discard events from
the queue")]
      HRESULT DiscardEvents(
         [in, defaultvalue(ALL_ENABLED_EVENTS)] EventType type,
         [in, defaultvalue(EVENT_ALL_MECH)] EventMechanism mech,
         [in, defaultvalue(0)] long customEventType);
      [helpcontext(HlpCtxIEventManager  + 5), helpstring("Wait for the specified
event")]
      HRESULT WaitOnEvent(
         [in] long waitTimeout,
         [in, defaultvalue(ALL_ENABLED_EVENTS)] EventType type,
         [in, defaultvalue(0)] long customEventType,
         [out, retval] IEvent **pEvent);
      [helpcontext(HlpCtxIEventManager  + 6), helpstring("Install a handler for
event callbacks")]
      HRESULT InstallHandler(
         [in] EventType type,
         [in] IEventHandler *handler,
         [in, defaultvalue(0)] long userHandle,
         [in, defaultvalue(0)] long customEventType);
      [helpcontext(HlpCtxIEventManager  + 7), helpstring("Remove a previously
installed handler")]
      HRESULT UninstallHandler(
         [in] EventType type,
         [in, defaultvalue(0)] long userHandle,
         [in, defaultvalue(0)] long customEventType);
   };
```

The `IEventManager` Interface has a COM property that corresponds to an attribute defined in VISA. The following table shows property-attribute equivalence for `IEventManager`.

Property Name	VISA Attribute Name
`MaximumQueueLength`	VI_ATTR_MAX_QUEUE_LENGTH

Table 3.3.1

The `IEventManager` interface has several methods that correspond to VISA functions. The following table shows method-function equivalence for `IEventManager`.

Method Name	VISA Equivalent
`EnableEvent`	`viEnableEvent`
`DisableEvent`	`viDisableEvent`
`DiscardEvents`	`viDiscardEvents`
`WaitOnEvent`	`viWaitOnEvent`
`InstallHandler`	`viInstallHandler`
`UninstallHandler`	`viUninstallHandler`

Table 3.3.2

RULE 3.3.1
All VISA COM I/O Resources **SHALL** implement IEventManager.

OBSERVATION 3.3.1
Since IEventManager is derived from IVisaSession, meeting RULE 3.3.1 also meets RULE 3.2.4.

RULE 3.3.2
The behaviors of these methods and properties for a particular resource **SHALL** be identical to section 3.7 of VPP 4.3 except where otherwise noted.

RULE 3.3.3
Instead of a function pointer, InstallHandler and UninstallHandler **SHALL** use a parameter of type IEventHandler. Resources that call methods on the callback function inside of IEventHandler **SHALL** follow all COM rules regarding safe COM method calls.

OBSERVATION 3.3.2
Since COM does not have the idea of function pointers, the proper way to connect callbacks in COM is through the use of object references (pointers to interfaces implemented by COM objects).

RECOMMENDATION 3.3.1
It is preferable that calls to the callback routine in IEventHandler not affect the liveness of the VISA COM I/O resource instance making the calls, i.e. they do not cause the resource to block.

RULE 3.3.4
If a call by a resource to the callback method of IEventHandler fails, the failure **SHALL** be ignored.

OBSERVATION 3.3.3
If a client is failing in its event handler routines and returning bad HRESULTs, it is likely that the client has become unstable and there is little that the resource can do to solve the problem or inform the user. Since it is a callback that is failing, sending another event indicating an error would probably also generate an error. The only other way of reporting the error, returning a predefined error code upon return of the next method call on the resource is undesirable because that means users who wished to capture all errors would have to watch for this error on every method of every resource, which is probably more trouble than it is worth.

OBSERVATION 3.3.4

Calling the `DisableEvent()` method prevents future events from being raised. When the implementation of `DisableEvent()` returns to the application, it is possible that a callback may still be active, possibly on another thread. It is valid for a user to invoke `DisableEvent()` from within a callback. It is not valid for a user to invoke `UninstallHandler()` from within a callback.

RECOMMENDATION 3.3.2

It is preferable that any implementation of `UninstallHandler()` should synchronize with all outstanding callbacks on the given session to ensure that the handler being removed is not in use.

3.3.2. IEvent Interface and the related event interfaces

The IEvent Interface represents the viEvent object in VISA. Additionally, there are specific interfaces for the various event types to allow direct access of event attributes as COM properties. Following is the IDL definition of `IEvent`.

```
   [
      object,
      oleautomation,
      helpstring("VISA Event Interface"),
      uuid(db8cbf12-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIEvent + 49),
      pointer_default(unique)
   ]
   interface IEvent : IUnknown
   {
      [propget, helpcontext(HlpCtxIEvent  + 1), helpstring("Get the event type")]
      HRESULT Type([out, retval] EventType *pVal);
      [propget, helpcontext(HlpCtxIEvent  + 2), helpstring("Get the custom event
type number")]
      HRESULT CustomEventTypeNumber([out, retval] long *pVal);

      [hidden, helpcontext(HlpCtxIEvent  + 3), helpstring("Get an attribute of
the event")]
      HRESULT GetAttribute(
         [in] long attribute,
         [out, retval] VARIANTARG *pAttrState);
      [hidden, helpcontext(HlpCtxIEvent  + 4), helpstring("Set an attribute of
the event")]
      HRESULT SetAttribute(
         [in] long attribute,
         [in] VARIANTARG attrState);
      [helpcontext(HlpCtxIEvent  + 5), helpstring("Close the event")]
      HRESULT Close();
   };
```

The `IEvent` Interface has a COM property that corresponds to an attribute defined in VISA. The following table shows property-attribute equivalence for `IEvent`.

Property Name	VISA Attribute Name
`Type`	VI_ATTR_EVENT_TYPE
`CustomEventTypeNumber`	VI_ATTR_EVENT_TYPE

Table 3.3.3

The `IEventManager` interface has several methods that correspond to VISA functions. The following table shows method-function equivalence for `IEventManager`.

Method Name	VISA Equivalent
`GetAttribute`	`viGetAttribute`
`SetAttribute`	`viSetAttribute`
`Close`	`viClose`

Table 3.3.4

Following are the definitions of the event-specific interfaces.

```
   [
      object,
      oleautomation,
      helpstring("I/O Completion Event Interface"),
      uuid(db8cbf15-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIEventIOCompletion + 49),
      pointer_default(unique)
   ]
   interface IEventIOCompletion : IEvent
   {
      [propget, helpcontext(HlpCtxIEventIOCompletion  + 1), helpstring("Get the
I/O status code of this transfer")]
      HRESULT IOStatus([out, retval] HRESULT *pVal);
      [propget, helpcontext(HlpCtxIEventIOCompletion  + 2), helpstring("Get the
job ID")]
      HRESULT JobId([out, retval] long *pVal);
      [propget, helpcontext(HlpCtxIEventIOCompletion  + 3), helpstring("Get the
number of elements transferred")]
      HRESULT ReturnCount([out, retval] long *pVal);
      [propget, helpcontext(HlpCtxIEventIOCompletion  + 4), helpstring("Get the
read buffer data")]
      HRESULT ReadBuffer([out, retval] SAFEARRAY(BYTE) *pVal);
      [propget, helpcontext(HlpCtxIEventIOCompletion  + 5), helpstring("Get the
read buffer as a string")]
      HRESULT ReadBufferAsString([out, retval] BSTR *pVal);
   };

   [
      object,
      oleautomation,
      helpstring("Trigger Event Interface"),
      uuid(db8cbf16-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIEventTrigger + 49),
      pointer_default(unique)
   ]
   interface IEventTrigger : IEvent
   {
      [propget, helpcontext(HlpCtxIEventTrigger  + 1), helpstring("Get the
trigger line on which this event was received")]
      HRESULT TriggerID([out, retval] TriggerLine *pVal);
   };

   [
      object,
      oleautomation,
      helpstring("VXI Signal Event Interface"),
      uuid(db8cbf17-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIEventVxiSignal + 49),
      pointer_default(unique)
   ]
   interface IEventVxiSignal : IEvent
   {
      [propget, helpcontext(HlpCtxIEventVxiSignal  + 1), helpstring("Get the 16-
bit signal Status/ID value")]
      HRESULT SignalStatusID([out, retval] short *pVal);
   };
```

```
[
     object,
     oleautomation,
     helpstring("VXI/VME Interrupt Event Interface"),
     uuid(db8cbf18-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxIEventVxiVmeInterrupt + 49),
     pointer_default(unique)
   ]
   interface IEventVxiVmeInterrupt : IEvent
   {
      [propget, helpcontext(HlpCtxIEventVxiVmeInterrupt  + 1), helpstring("Get
the 32-bit interrupt Status/ID value")]
     HRESULT InterruptStatusID([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIEventVxiVmeInterrupt  + 2), helpstring("Get
the interrupt level on which this event was received")]
     HRESULT InterruptLevel([out, retval] short *pVal);
   };

   [
     object,
     oleautomation,
     helpstring("GPIB CIC Event Interface"),
     uuid(db8cbf19-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxIEventGpibCIC + 49),
     pointer_default(unique)
   ]
   interface IEventGpibCIC : IEvent
   {
      [propget, helpcontext(HlpCtxIEventGpibCIC  + 1), helpstring("Get the
controller CIC state")]
     HRESULT CICState([out, retval] VARIANT_BOOL *pVal);
   };

   [
     object,
     oleautomation,
     helpstring("USB Interrupt Event Interface"),
     uuid(DB8CBF23-D6D3-11D4-AA51-00A024EE30BD),
     helpcontext(HlpCtxIEventUsbInterrupt + 49),
     pointer_default(unique)
   ]
   interface IEventUsbInterrupt : IEvent
   {
      [propget, helpcontext(HlpCtxIEventUsbInterrupt  + 1), helpstring("Get the
received buffer data")]
     HRESULT DataBuffer([out, retval] SAFEARRAY(BYTE) *pVal);
     [propget, helpcontext(HlpCtxIEventUsbInterrupt  + 2), helpstring("Get the
I/O status code of this transfer")]
     HRESULT IOStatus([out, retval] HRESULT *pVal);
     [propget, helpcontext(HlpCtxIEventUsbInterrupt  + 3), helpstring("Get the
actual number of bytes received")]
     HRESULT InterruptSize([out, retval] short *pVal);
       };
```

These interfaces have several COM properties that correspond to VISA attributes.

Property Name	VISA Attribute Name
IOStatus	VI_ATTR_STATUS
JobId	VI_ATTR_JOB_ID
ReturnCount	VI_ATTR_RET_COUNT
InterruptStatusID	VI_ATTR_INTR_STATUS_ID
InterruptLevel	VI_ATTR_RECV_INTR_LEVEL
ReadBuffer	VI_ATTR_BUFFER
ReadBufferAsString	VI_ATTR_BUFFER
SignalStatusID	VI_ATTR_SIGP_STATUS_ID
CICState	VI_ATTR_GPIB_RECV_CIC_STATE
TriggerID	VI_ATTR_RECV_TRIG_ID
DataBuffer	VI_ATTR_USB_RECV_INTR_DATA
InterruptSize	VI_ATTR_USB_RECV_INTR_SIZE

Table 3.3.5

RULE 3.3.5
The IOStatus property's output parameter **SHALL** contain the HRESULT that would have been generated had the method been synchronous.

RULE 3.3.6
The ReadBufferAsString property **SHALL** convert the result into a BSTR using standard conversion from ASCII and return the BSTR.

RULE 3.3.7
The system resources associated with an event object **SHALL** be released when the reference count for the event object goes to zero.

RULE 3.3.8
The event object associated with the IEvent interface **SHALL** have a reference count of one when passed to the client either through WaitOnEvent or the IEventHandler callback.

RULE 3.3.9
The event object **SHALL** implement the proper event-specific interface based on the event type.

Event Type	Event-specific Interface
VI_EVENT_TRIG	IEventTrigger
VI_EVENT_IO_COMPLETION	IEventIOCompletion
VI_EVENT_VXI_VME_INTR	IEventVxiVmeInterrupt
VI_EVENT_VXI_SIGP	IEventVxiSignal
VI_EVENT_GPIB_CIC	IEventGpibCIC
VI_EVENT_USB_INTR	IEventUsbInterrupt

Table 3.3.6

OBSERVATION 3.3.5
Because sharing memory between components violates the rules of COM, the only time a user will have access to the data returned from an asynchronous read is when they receive the I/O Completion event. At

that point, they receive a buffer containing the data that was read (if any) if they query the ReadBuffer or ReadBufferAsString property.

OBSERVATION 3.3.6
Other event types defined by VISA are accessible via the base IEvent interface.

3.3.3. IEventHandler Interface

This interface contains one method implemented by the user of a VISA COM I/O resource instance. The method `HandleEvent` on the interface is called by the resource when an event enabled to be asynchronously delivered occurs.

```
   [
      object,
      oleautomation,
      helpstring("User-implemented Event Handler Interface"),
      uuid(db8cbf13-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIEventHandler + 49),
      pointer_default(unique)
   ]
   interface IEventHandler : IUnknown
   {
      [helpcontext(HlpCtxIEventHandler  + 1), helpstring("User-implemented event
handler")]
      HRESULT HandleEvent(
         [in] IEventManager *vi,
         [in] IEvent *event,
         [in] long userHandle);
   };
```

OBSERVATION 3.3.7
If VISA COM I/O users who implement this interface block inside this method, the thread used to handle this event will not return until the block completes. Component developers should not assume an event thread will return in any timeframe.

RULE 3.3.10
VISA COM I/O resource components **SHALL NOT** kill blocked event threads except during process termination.

RULE 3.3.11
VISA COM I/O resource components **SHALL** continue to be responsive and operate while one or more event threads are blocked.

RULE 3.3.12
VISA COM I/O resource components **SHALL NOT** simultaneously deliver more than one event per IEventManager instance.

OBSERVATION 3.3.8
VISA COM I/O Users who implement this interface could return an error HRESULT or put an IErrorInfo structure on the thread-local storage.

RULE 3.3.13
VISA COM I/O resource components **SHALL** remove any error information from the thread-local storage **AND SHALL** ignore the HRESULTs returned from the HandleEvent method.

OBSERVATION 3.3.9
Users that implement this method may use any COM apartment type. In Visual Basic, it is likely to be in an STA.

OBSERVATION 3.3.10
If the component that implements this method is in a STA (single-threaded apartment), the COM system will serialize calls to this method, meeting the behavior defined in RULE 3.3.12.

OBSERVATION 3.3.11

If a user component that implements this method is in an STA, this event will not be executed until the component makes a COM method call that leaves the component's apartment, or until the user component enters the windows message loop.

Section 4: VISA COM I/O Resource Management

This section describes the mechanisms available in VISA COM I/O to control and manage resources. This includes, but is not limited to, the assignment of unique resource addresses, and unique resource IDs. This work is split between a Global Resource Manager and vendor-specific resource managers.

The Global Resource Manager can create any resource in a VISA COM I/O system. It gives users of VISA COM I/O access to individual resources and provides the services described below. The VISA equivalent of this component is the VISA Resource Manager. The VISA COM I/O resource management scheme allows extensibility of the VISA COM I/O system to support new resources by a loose coupling between the Global resource manager and the Vendor-specific resource managers, and the ability to dynamically discover new Vendor managers and upgraded abilities of existing managers. Additionally, the Global Resource Manager and the Vendor-Specific Resource Managers share the same interface, which allows for polymorphic resource management between global and vendor-specific resources.

The VISA Resource Manager resource provides basic services to applications that include searching for resources, and the ability to open sessions to these resources. A summary of these services for VISA is presented below:

- **Access**

  The Global Resource Manager allows the opening of sessions to resources established on request by applications. Users can request this service using `Open()` on the IResourceManager interface of the Global Resource Manager component. The system has responsibility of freeing up all the associated system resources whenever an application closes the session or becomes dysfunctional.

- **Search**

  These services are used to find a resource in order to establish a communication link to it. The search is based on a description string. The VISA COM I/O Global Resource Manager delegates search responsibility to all of the Vendor-Specific Resource Managers. Instead of locating and searching for individual resources, the Vendor-Specific Resource Manager searches for resources associated with an I/O interface. The Global Resource Manager is responsible for resolving search conflicts: situations where more than one Vendor-Specific Resource Manager returns a particular resource string in response to a search. Users can request this service by using the `FindRsrc` method on the `IResourceManager` interface.

4.1. IResourceManager Interfaces

The behavior of some of the methods of the `IResourceManager` interfaces depends on whether the Global Resource Manager component or a Vendor-Specific Resource Manager component is implementing it. Those methods will be described twice in Section 4, *VISA COM I/O Resource Management*, once for the global case and once for the vendor-specific case. Below is the IDL definition of the `IResourceManager` interfaces.

```
   [
      object,
      oleautomation,
      helpstring("VISA Resource Manager Interface (obsolete)"),
      uuid(db8cbf02-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIResourceManager + 49),
      pointer_default(unique),
      hidden
   ]
   interface IResourceManager : IUnknown
   {
      [propget,helpstring("Get the manufacturer name of the
component"),helpcontext(HlpCtxIResourceManager + 1)]
      HRESULT SoftwareManufacturerName([out, retval] BSTR *pVal);
      [propget,helpstring("Get the manufacturer ID of the
component"),helpcontext(HlpCtxIResourceManager + 2)]
      HRESULT SoftwareManufacturerID([out, retval] short *pVal);
      [propget,helpstring("Get the description of the
component"),helpcontext(HlpCtxIResourceManager + 3)]
      HRESULT Description([out, retval] BSTR *pDesc);
      [propget,helpstring("Get the implementation version of the
component"),helpcontext(HlpCtxIResourceManager + 4)]
      HRESULT ComponentVersion([out, retval] long *pVal);
      [propget,helpstring("Get the ProgID of the
component"),helpcontext(HlpCtxIResourceManager + 5)]
      HRESULT ProgID([out, retval] BSTR *pVal);
      [propget,helpstring("Get the VISA COM I/O specification
version"),helpcontext(HlpCtxIResourceManager + 6)]
      HRESULT SpecVersion([out, retval] long *pVal);

      [helpstring("Find a list of resources that match a search
string"),helpcontext(HlpCtxIResourceManager + 7)]
      HRESULT FindRsrc(
         [in] BSTR expr,
         [out, retval] SAFEARRAY(BSTR) *pFindList);
      [helpstring("Initialize a session to the specified resource
name"),helpcontext(HlpCtxIResourceManager + 9)]
      HRESULT Open(
         [in] BSTR resourceName,
         [in, defaultvalue(NO_LOCK)] AccessMode mode,
         [in, defaultvalue(2000)] long openTimeout,
         [in, defaultvalue("")] BSTR optionString,
         [out, retval] IVisaSession **vi);
      [helpstring("Determine the validity and interface information of a resource
name"),helpcontext(HlpCtxIResourceManager + 10)]
      HRESULT ParseRsrc(
         [in] BSTR resourceName,
         [in, out] short *pInterfaceType,
         [in, out] short *pInterfaceNumber,
         [in, out] BSTR *pSessionType);
   };
```

```
   [
      object,
      oleautomation,
      helpstring("VISA Resource Manager Interface"),
      uuid(db8cbf20-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIResourceManager3 + 49),
      pointer_default(unique)
   ]
   interface IResourceManager3 : IResourceManager
   {
      [helpstring("Determine the validity and interface information of a resource
name"),helpcontext(HlpCtxIResourceManager3 + 1)]
      HRESULT ParseRsrcEx(
         [in] BSTR resourceName,
         [in, out] short *pInterfaceType,
         [in, out] short *pInterfaceNumber,
         [in, out] BSTR *pSessionType,
         [in, out] BSTR *pUnaliasedExpandedResourceName,
         [in, out] BSTR *pAliasIfExists);
   };
```

RULE 4.1.1

The `SpecVersion` property for the global resource manager component **SHALL** return the specification version of VISA COM I/O as defined in VISA by the `VI_ATTR_RSRC_SPEC_VERSION` property.

RULE 4.1.2

The `SpecVersion` property for the vendor-specific resource manager component **SHALL** return the specification version of VISA COM I/O as defined in VISA by the `VI_ATTR_RSRC_SPEC_VERSION` property.

4.2. The Vendor-Specific Resource Manager Component

RULE 4.2.1
A vendor-specific resource manager component **SHALL** be able to create instances of one or more resource COM components provided by that vendor.

RULE 4.2.2
There **SHALL** be only one vendor-specific resource manager for a particular resource COM component.

RULE 4.2.3
The `SoftwareManufacturerName` and `SoftwareManufacturerID` properties of a vendor-specific component **SHALL** be identical to the `SoftwareManufacturerName` and `SoftwareManufacturerID` properties of the resources it creates.

PERMISSION 4.2.1
The vendor-specific version of the `Description` property **MAY** be implemented as the vendor sees fit.

RULE 4.2.4
The vendor-specific version of the `Description` property **SHALL** always return the same string.

RULE 4.2.5
The `ProgID` property of a vendor-specific manager **SHALL** return the exact string in the win32 registry that can be used to create the component.

RULE 4.2.6
The `ComponentVersion` property of the vendor-specific manager **SHALL** behave identically to the VISA attribute `VI_ATTR_RSRC_IMPL_VERSION`.

RULE 4.2.7
The FindRsrc method of the vendor-specific manager **SHALL** return a SAFEARRAY containing one or more BSTRs containing valid VISA resource strings.

RULE 4.2.8
The behavior of FindRsrc **SHALL** be identical to a call in VISA of viFindRsrc followed by viFindNext until all discovered resources are found.

OBSERVATION 4.2.1
Unlike VISA's viFindRsrc, the vendor-specific version of FindRsrc is only responsible for finding resources supported by the vendor's VISA COM I/O resource components.

RULE 4.2.9
The vendor-specific ParseRsrc method **SHALL** have the same behavior as the viParseRsrc method described in VPP4.3 with the following exceptions.

RULE 4.2.10
The vendor-specific ParseRsrc **SHALL** understand resource strings only for interface types, session types, and interface numbers that it supports.

RULE 4.2.11
The ParseRsrc method **SHALL** NOT perform any I/O

RULE 4.2.12
IF a vendor-specific resource manager can create any particular resource on a hardware interface, **THEN** it **SHALL** be capable of creating all available resources on that interface. Availability of a resource is defined in VPP 4.3. See Section 5, *VISA COM I/O Resource Classes*, for specific rules regarding requirements.

RULE 4.2.13
A vendor-specific resource manager **SHALL** implement the COM interface `IProvideClassInfo2`.

RULE 4.2.14

A vendor-specific resource manager that complies with the VISA 3.0 specification **SHALL** implement the IResourceManager3 interface.

RULE 4.2.15

The ParseRsrcEx method of vendor-specific resource managers **SHALL** behave identically to the ParseRsrc method **AND SHALL** provide the extra out parameters as described in the VISA 4.3 specification for viParseRsrcEx.

RULE 4.2.16

The GRM's ParseRsrcEx **SHALL** return S_VISA_EXT_FUNC_NIMPL if no vendor-specific resource manager exists that supports IResourceManager3 or if the only vendor-specific resource manager(s) that returns a success value for ParseRsrc for the given resource string does not support IResourceManager3. The method **SHALL** return the appropriate parse error code if no SRM ParseRsrc or ParseRsrcEx returns a success code.

OBSERVATION 4.2.2

The RULE 4.2.16 compels the GRM to call ParseRsrc during a ParseRsrcEx call on each SRM that does not support IResourceManager3 if no SRM ParseRsrcEx implementation can parse the resource string in order to better determine if the string is unparsable or if the compatible SRM does not support ParseRsrcEx. This is for better backward compatibility between multiple modules of potentially different versions.

4.3. The Global Resource Manager Component

The Global Resource Manager's main responsibilities are locating, instantiating, and using the vendor managers and resolving any overlapping functionality between vendor-specific managers. It is distributed with the VISA COM I/O type library.

4.3.1. The Global Component Implementation

RULE 4.3.1
The SoftwareManufacturerName property **SHALL** return "`VXIplug&play Alliance`" and the SoftwareManufacturerID property **SHALL** return 0x3FFF.

RULE 4.3.2
The description property **SHALL** return "Global VISA COM I/O Resource Manager".

RULE 4.3.3
The ComponentVersion and SpecVersion properties **SHALL** follow the rules of ViVersion.

RULE 4.3.4
The ProgID property **SHALL** return "`VISA.GlobalRM`".

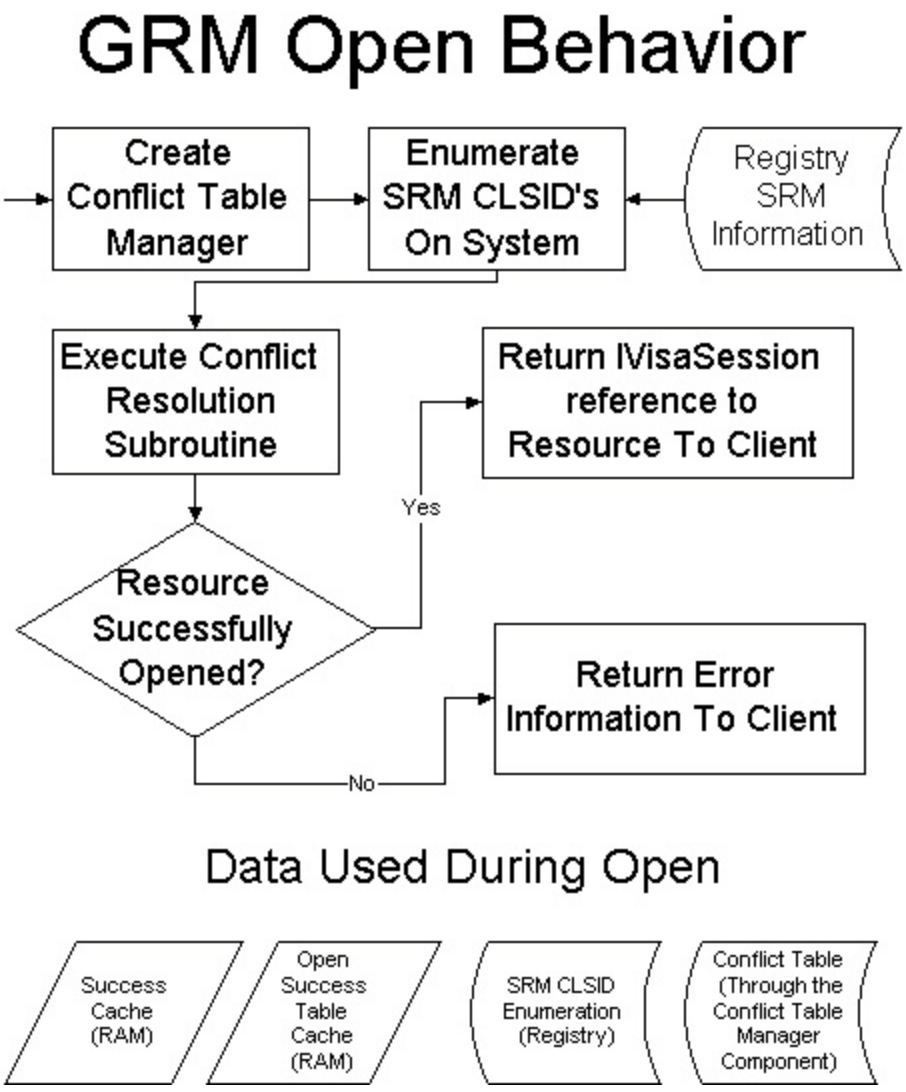
RULE 4.3.5
The FindRsrc method **SHALL** call the FindRsrc method on all the vendor-specific resource managers. Any resource strings that are equivalent according to the rules defined in VPP 4.3 for resource strings **SHALL** be discarded, and a new SAFEARRAY with the combined results **SHALL** be returned to the user.

RULE 4.3.6
The Open method of the Global Resource Manager **SHALL** behave as described in the following diagram.

RULE 4.3.7
The global resource manager **SHALL** implement the COM interface `IProvideClassInfo2`.

RULE 4.3.8
The version resource of the global resource manager DLL implementing the VISA 3.0 specification and the IResourceManager3 interface shall be higher than the version implementing the VISA 2.2 specification.

See VPP-4.3.5 for additional details about the Global Resource Manager implementation.


GRM Open Behavior
Create Conflict Table Manager
Enumerate SRM CLSID's On System
Registry SRM Information
Execute Conflict Resolution Subroutine
Return IVisaSession reference to Resource To Client
Yes
Resource Successfully Opened?
Return Error Information To Client
No
Data Used During Open
Success Cache (RAM)
Open Success Table Cache (RAM)
SRM CLSID Enumeration (Registry)
Conflict Table (Through the Conflict Table Manager Component)

4.4. The VISA Resource Conflict Manager Interface

See VPP-4.3.5 for details about the Conflict Resolution Manager implementation.

The global resource manager uses an implementation of the `IVisaConflictTableManager` interface to resolve conflicts where multiple VISA COM I/O implementations support the same resource. Below is the IDL definition of the `IVisaConflictTableManager` interface:

```
 [
   object,
   oleautomation,
   helpstring("VISA Resource Conflict Manager Interface"),
   uuid(db8cbf1b-d6d3-11d4-aa51-00a024ee30bd),
   helpcontext(HlpCtxIConflictManager + 49),
   pointer_default(unique),
   hidden
 ]
 interface IVisaConflictTableManager : IUnknown
 {
   typedef [public, helpstring("GUID Handler Types")]
   enum ConflictHandlerType {
      NotChosen,
      ChosenByResourceManager,
      ChosenByUser
   } ConflictHandlerType;

   [propget,helpstring("Get/Set whether to store just conflicts or all
resources"),helpcontext(HlpCtxIConflictManager + 1)]
   HRESULT StoreConflictsOnly([out, retval] VARIANT_BOOL *pVal);
   [propput,helpstring("Get/Set whether to store just conflicts or all
resources"),helpcontext(HlpCtxIConflictManager + 1)]
   HRESULT StoreConflictsOnly([in] VARIANT_BOOL newVal);
   [propget,helpstring("Get the filename of the conflict
table"),helpcontext(HlpCtxIConflictManager + 2)]
   HRESULT ConflictTableFilename([out, retval] BSTR *pVal);
   [propget,helpstring("Get the number of resource entries in the
table"),helpcontext(HlpCtxIConflictManager + 3)]
   HRESULT NumberOfResources([out, retval] long *pVal);

   [helpstring("Add or update a handler in the
table"),helpcontext(HlpCtxIConflictManager + 4)]
   HRESULT CreateHandler(
      [in] short interfaceType,
      [in] short interfaceNumber,
      [in] BSTR sessionType,
      [in] BSTR vsrmGuid,
      [in] ConflictHandlerType type,
      [in, defaultvalue("")] BSTR miscComments);
   [helpstring("Remove a specific handler from the
table"),helpcontext(HlpCtxIConflictManager + 5)]
   HRESULT DeleteHandler(
      [in] short interfaceType,
      [in] short interfaceNumber,
      [in] BSTR sessionType,
      [in] BSTR vsrmGuid);
   [helpstring("Remove all non-user-specified handlers for a given
GUID"),helpcontext(HlpCtxIConflictManager + 6)]
   HRESULT DeleteHandlerByGUID(
       [in] BSTR vsrmGuid);
```

```
    [helpstring("Remove a resource entry from the
 table"),helpcontext(HlpCtxIConflictManager + 7)]
    HRESULT DeleteResourceByIndex(
        [in] long tableIndex);
    [helpstring("Find the specified handler for a given
 resource"),helpcontext(HlpCtxIConflictManager + 8)]
    HRESULT FindChosenHandler(
       [in] short interfaceType,
       [in] short interfaceNumber,
       [in] BSTR sessionType,
       [in, out] BSTR *pVsrmGuid,
       [in, out] ConflictHandlerType *pType);
    [helpstring("Get the resource information for a given
 index"),helpcontext(HlpCtxIConflictManager + 9)]
    HRESULT QueryResource(
       [in] long tableIndex,
       [in, out] short *pInterfaceType,
       [in, out] short *pInterfaceNumber,
       [in, out] BSTR *pSessionType,
       [in, out] short *pNumHandlers);
    [helpstring("Get the handler information for a given
 resource"),helpcontext(HlpCtxIConflictManager + 10)]
    HRESULT QueryResourceHandler(
       [in] long tableIndex,
       [in] short handlerIndex,
       [in, out] BSTR *pVsrmGuid,
       [in, out] ConflictHandlerType *pType,
       [in, out] BSTR *pMiscComments);
    [helpstring("Save any changes"),helpcontext(HlpCtxIConflictManager + 11)]
    HRESULT FlushToFile();
};
```

OBSERVATION 4.4.1

Users should not assume an object that supports `IResourceManager` can be QueryInterface'd for `IVisaConflictTableManager` and vice-versa.

OBSERVATION 4.4.2

Users should not need to use `IVisaConflictTableManager` directly. It is documented in this specification to guarantee compatibility across implementations and for use by the global `IResourceManager` and by external utilities.

Section 5: VISA COM I/O Resource Classes

VISA COM I/O provides a subset of the most commonly used resource classes defined in VPP 4.3. Because of the built-in extensibility of COM, there is the potential to provide other resource classes that behave like the predefined classes, and they will work with the VISA COM I/O libraries.

RULE 5.0.1

Any resource component which implements a predefined resource type **SHALL** return the predefined interface type number and name for that interface from the IVisaSession interface.

RULE 5.0.2

Any resource component which implements a non-pre-defined resource type **SHALL** return an interface type number of 0x5000-0x6FFF and the name **SHALL NOT** match that of any predefined interface or reserved name.

RECOMMENDATION 5.0.1

For non-VISA-defined resource types, a manufacturer should include the manufacturer name in the interface name to avoid confuision or possible conflicts with future VISA-defined resource types.

5.1. INSTR Resources

Resources of this type provide either basic stream I/O to instruments as laid out by IEEE 488.2 or register operations or both. See VPP4.3 section 5.1 for more information about these resources. The functionality of INSTR resources is broken up into several COM interfaces in VISA COM I/O. Users can write code that polymorphically acts on any INSTR resource type by using only these resources and the Init string to create, instantiate, and use instruments. For register-based resources, it should be noted here that no address mapping or window services are provided in VISA COM I/O because of limitations of the COM calling conventions necessary to provide remote method invocation functionality.

RULE 5.1.1
All VISA COM I/O resources that implement the GPIB, TCPIP, VXI, GPIB-VXI, and ASRL INSTR resources **SHALL** implement the interfaces IBaseMessage, IMessage, and IAsyncMessage.

RULE 5.1.2
All VISA COM I/O resources that implement the VXI and GPIB-VXI INSTR resources **SHALL** implement the interfaces IRegister and ISharedRegister.

RULE 5.1.3
All VISA COM I/O resources that implement the GPIB and GPIB-VXI INSTR resources **SHALL** implement the interface IGpib.

RULE 5.1.4
All VISA COM I/O resources that implement the VXI and GPIB-VXI INSTR resources **SHALL** implement the interface IVxi.

RULE 5.1.5
IF a VISA COM I/O resource implements the VXI or GPIB-VXI INSTR resource **AND** it complies with the VISA 3.0 specification, **THEN** it **SHALL** implement the interface IVxi3.

RULE 5.1.6
All VISA COM I/O resources that implement the ASRL INSTR resource **SHALL** implement the interface ISerial.

RULE 5.1.7
All VISA COM I/O resources that implement the TCPIP INSTR resource **SHALL** implement the interface ITcpipInstr.

RULE 5.1.8
All VISA COM I/O resources that implement the USB INSTR resource **SHALL** implement the interface IUsb.

RULE 5.1.9
INSTR VISA COM I/O resources **SHALL** return E_NOINTERFACE when QueryInterface'd for an interface defined by VISA COM I/O other than the ones explicitly required or allowed to be implemented.

5.1.1. IBaseMessage Interface

The IBaseMessage interface provides the methods and properties for stream reading/writing except for the methods specific to asynchronous, or regular I/O. The IMessage and IAsyncMessage interfaces supply those specific methods and derive from IBaseMessage. Following is the IDL specification for the IBaseMessage interface.

```
[
   object,
   oleautomation,
   helpstring("IBaseMessage - do not use directly"),
   uuid(db8cbf04-d6d3-11d4-aa51-00a024ee30bd),
   helpcontext(HlpCtxIBaseMessage + 49),
   pointer_default(unique),
   hidden
]
interface IBaseMessage : IVisaSession
```

```
   {
      [propget, helpcontext(HlpCtxIBaseMessage  + 1), helpstring("Get/Set which
I/O protocol to use")]
     HRESULT IOProtocol([out, retval] IOProtocol *pVal);
     [propput, helpcontext(HlpCtxIBaseMessage  + 1), helpstring("Get/Set which
I/O protocol to use")]
     HRESULT IOProtocol([in] IOProtocol newVal);
     [propget, helpcontext(HlpCtxIBaseMessage  + 2), helpstring("Get/Set whether
to assert END on Write")]
     HRESULT SendEndEnabled([out, retval] VARIANT_BOOL *pVal);
     [propput, helpcontext(HlpCtxIBaseMessage  + 2), helpstring("Get/Set whether
to assert END on Write")]
     HRESULT SendEndEnabled([in] VARIANT_BOOL newVal);
     [propget, helpcontext(HlpCtxIBaseMessage  + 3), helpstring("Get/Set the
termination character")]
     HRESULT TerminationCharacter([out, retval] BYTE *pVal);
     [propput, helpcontext(HlpCtxIBaseMessage  + 3), helpstring("Get/Set the
termination character")]
     HRESULT TerminationCharacter([in] BYTE newVal);
     [propget, helpcontext(HlpCtxIBaseMessage  + 4), helpstring("Get/Set whether
to use the termination character on Read")]
     HRESULT TerminationCharacterEnabled([out, retval] VARIANT_BOOL *pVal);
     [propput, helpcontext(HlpCtxIBaseMessage  + 4), helpstring("Get/Set whether
to use the termination character on Read")]
     HRESULT TerminationCharacterEnabled([in] VARIANT_BOOL newVal);

     [helpcontext(HlpCtxIBaseMessage  + 5), helpstring("Assert a trigger")]
     HRESULT AssertTrigger(
         [in, defaultvalue(TRIG_PROT_DEFAULT)] TriggerProtocol protocol);
     [helpcontext(HlpCtxIBaseMessage  + 6), helpstring("Clear the device")]
     HRESULT Clear();
     [helpcontext(HlpCtxIBaseMessage  + 7), helpstring("Read the status byte")]
     HRESULT ReadSTB(
         [out, retval] short *pStatusByte);
   };
```

RULE 5.1.10

VISA COM I/O resources **SHALL** implement these methods as specified in VPP 4.3 except where specified otherwise in this specification.

5.1.2. IMessage Interface

This interface provides unbuffered synchronous stream communications with an instrument resource. Below is the IDL specification for the IMessage interface.

```
   [
      object,
      oleautomation,
      helpstring("Message Based Interface"),
      uuid(db8cbf05-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIMessage + 49),
      pointer_default(unique)
   ]
   interface IMessage : IBaseMessage
   {
      [helpcontext(HlpCtxIMessage  + 1), helpstring("Read the specified number
of bytes")]
     HRESULT Read(
        [in] long count,
        [out, retval] SAFEARRAY(BYTE) *pBuffer);
     [helpcontext(HlpCtxIMessage  + 2), helpstring("Read the specified number
of bytes as a string")]
     HRESULT ReadString(
        [in] long count,
        [out, retval] BSTR *pBuffer);
     [helpcontext(HlpCtxIMessage  + 3), helpstring("Write the specified data")]
```

```
     HRESULT Write(
        [in] SAFEARRAY(BYTE) *buffer,
        [in] long count,
        [out, retval] long *pRetCount);
     [helpcontext(HlpCtxIMessage  + 4), helpstring("Write the specified
string")]
     HRESULT WriteString(
        [in] BSTR buffer,
        [out, retval] long *pRetCount);
   };
```

Below is a table showing the methods of the IMessage Interface and their equivalents in the VISA API.

IMessage Method	VISA Function
Read	viRead
Write	viWrite
ReadString	viRead
WriteString	viWrite

Table 5.1.1

RULE 5.1.11
Unless otherwise noted, the methods of IMessage **SHALL** behave identically to their equivalents in VISA.

RULE 5.1.12
Both Read and Write **SHALL** use SAFEARRAYs of unsigned characters to retrieve and send stream data.

RULE 5.1.13
The Write method **SHALL** return the HRESULT E_INVALIDARG or the equivalent VISA HRESULT if the parameter count is larger than the size of the SAFEARRAY passed in.

RECOMMENDATION 5.1.1
It is recommended that upon an invalid count parameter, there should be an IErrorInfo structure placed on the thread-local storage that describes the error more specifically.

RULE 5.1.14
If the Write method is called with the parameter count smaller than the size of the SAFEARRAY passed in, only the first count bytes **SHALL** be written to the instrument resource.

RULE 5.1.15
The status parameter **SHALL** equal the return value used by viRead and viWrite in VISA upon the return of the methods Read and Write.

OBSERVATION 5.1.1
Although COM APIs, like C APIs can return errors as the return value of functions/methods, many COM environments have problems understanding or ignore return values that are successful other than S_OK, so successful return values that indicate various success conditions are not feasible in COM.

RULE 5.1.16
The ReadString and WriteString methods **SHALL** behave identically to the Read and Write methods but will give and receive BSTRs instead of SAFEARRAYs of BYTEs.

RULE 5.1.17
WriteString **SHALL**fail with the error code E_VISA_INV_FMT when one or more of the Unicode characters in the Message argument have an ambiguous or no valid conversion to ASCII.

5.1.3. IAsyncMessage Interface

The IAsyncMessage interface implements the methods viReadAsync and viWriteAsync for instrument resources. Additionally, it provides the equivalent of the VISA Template function viTerminate. Below is the IDL specification for IAsyncMessage.

```
   [
      object,
      oleautomation,
      helpstring("Asynchronous Message Based Interface"),
      uuid(db8cbf06-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIAsyncMessage + 49),
      pointer_default(unique)
   ]
   interface IAsyncMessage : IBaseMessage
   {
      [helpcontext(HlpCtxIAsyncMessage  + 1), helpstring("Read the specified
number of bytes")]
      HRESULT Read(
         [in] long count,
         [out, retval] long *pJobId);
      [helpcontext(HlpCtxIAsyncMessage  + 2), helpstring("Write the specified
data")]
      HRESULT Write(
         [in] SAFEARRAY(BYTE) *Buffer,
         [in] long count,
         [out, retval] long *pJobId);
      [helpcontext(HlpCtxIAsyncMessage  + 3), helpstring("Write the specified
string")]
      HRESULT WriteString(
         [in] BSTR buffer,
         [out, retval] long *pJobId);
      [helpcontext(HlpCtxIAsyncMessage  + 4), helpstring("Terminate the specified
asynchronous job")]
      HRESULT Terminate(
          [in] long jobId);
   };
```

The following table shows the methods of IAsyncMessage and their equivalents in the VISA API.

IAsyncMessage Method	VISA Function
Read	viReadAsync
WriteString	viWriteAsync
Write	viWriteAsync
Terminate	viTerminate

Table 5.1.2

RULE 5.1.18
Unless otherwise noted, the methods of IAsyncMessage **SHALL** behave identically to their equivalents in VISA.

RULE 5.1.19
Write **SHALL** use a SAFEARRAY of unsigned characters to send stream data.

RULE 5.1.20
There **SHALL NOT** be a buffer for the Read method to place data in while the asynchronous call completes.

OBSERVATION 5.1.2
As noted in the events section, the rules of COM prohibit shared memory between COM components and their clients. The only time data from an asynchronous read data is available is during the I/O completion event.

OBSERVATION 5.1.3
Unlike the VISA API, the Terminate method is in the IAsyncMessage interface, which is part of the Instrument Control Resource API. In VISA, the Terminate method is part of the resource template, and

therefore part of all resources, including instrument resource template. Since the only asynchronous jobs defined in VISA are asynchronous reads and writes, this is desirable.

RULE 5.1.21

The Write method **SHALL** return the HRESULT E_INVALIDARG if the parameter count is larger than the size of the SAFEARRAY passed in.

RECOMMENDATION 5.1.2

It is recommended that upon an invalid count parameter, there should be an IErrorInfo structure placed on the thread-local storage that describes the error more specifically.

RULE 5.1.22

If the Write method is called with the parameter count smaller than the size of the SAFEARRAY passed in, only the first count bytes **SHALL** be written to the instrument resource.

OBSERVATION 5.1.4

Even if the Read operation is implemented synchronously, the only opportunity to retrieve the buffer is still through the I/O completion event.

PERMISSION 5.1.1

Instrument Control Resources that implement the IAsyncMessage interface synchronously may call the I/O completion event as a reentrant callback.

RULE 5.1.23

The WriteString method **SHALL** behave identically to the Write method but WriteString will send a BSTR instead of a SAFEARRAY of BYTEs.

RULE 5.1.24

WriteString **SHALL** fail with the error code E_VISA_INV_FMT when one or more of the Unicode characters in the Message argument have an ambiguous or no valid conversion to ASCII.

5.1.4. IRegister Interface

The IRegister interface provides a means of register access for INSTR session types such as VXI. Below is the IDL specification for the IRegister interface.

```
   [
      object,
      oleautomation,
      helpstring("Register Based Interface"),
      uuid(db8cbf07-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIRegister + 49),
      pointer_default(unique)
   ]
   interface IRegister : IVisaSession
   {
      [propget, helpcontext(HlpCtxIRegister  + 1), helpstring("Get/Set whether
the target format is Big Endian")]
      HRESULT DestinationBigEndian([out, retval] VARIANT_BOOL *pVal);
      [propput, helpcontext(HlpCtxIRegister  + 1), helpstring("Get/Set whether
the target format is Big Endian")]
      HRESULT DestinationBigEndian([in] VARIANT_BOOL newVal);
      [propget, helpcontext(HlpCtxIRegister  + 2), helpstring("Get/Set the target
increment on Move")]
      HRESULT DestinationIncrement([out, retval] long *pVal);
      [propput, helpcontext(HlpCtxIRegister  + 2), helpstring("Get/Set the target
increment on Move")]
      HRESULT DestinationIncrement([in] long newVal);
      [propget, helpcontext(HlpCtxIRegister  + 3), helpstring("Get/Set whether
the source format is Big Endian")]
      HRESULT SourceBigEndian([out, retval] VARIANT_BOOL *pVal);
      [propput, helpcontext(HlpCtxIRegister  + 3), helpstring("Get/Set whether
the source format is Big Endian")]
      HRESULT SourceBigEndian([in] VARIANT_BOOL newVal);
      [propget, helpcontext(HlpCtxIRegister  + 4), helpstring("Get/Set the source
increment on Move")]
```

```
     HRESULT SourceIncrement([out, retval] long *pVal);
     [propput, helpcontext(HlpCtxIRegister  + 4), helpstring("Get/Set the source
increment on Move")]
     HRESULT SourceIncrement([in] long newVal);

     [helpcontext(HlpCtxIRegister  + 5), helpstring("Read a value from the
memory location")]
     HRESULT In8(
        [in] short space,
        [in] long offset,
        [out, retval] BYTE *pVal8);
     [helpcontext(HlpCtxIRegister  + 6), helpstring("Read a value from the
memory location")]
     HRESULT In16(
        [in] short space,
        [in] long offset,
        [out, retval] short *pVal16);
     [helpcontext(HlpCtxIRegister  + 7), helpstring("Read a value from the
memory location")]
     HRESULT In32(
        [in] short space,
        [in] long offset,
        [out, retval] long *pVal32);
     [helpcontext(HlpCtxIRegister  + 8), helpstring("Write a value to the memory
location")]
     HRESULT Out8(
        [in] short space,
        [in] long offset,
        [in] BYTE val8);
     [helpcontext(HlpCtxIRegister  + 9), helpstring("Write a value to the memory
location")]
     HRESULT Out16(
        [in] short space,
        [in] long offset,
        [in] short val16);
     [helpcontext(HlpCtxIRegister  + 10), helpstring("Write a value to the
memory location")]
     HRESULT Out32(
        [in] short space,
        [in] long offset,
        [in] long val32);
     [helpcontext(HlpCtxIRegister  + 11), helpstring("Read data from the memory
location")]
     HRESULT MoveIn8(
        [in] short space,
        [in] long offset,
        [in] long length,
        [out, retval] SAFEARRAY(BYTE) *pBuf8);
     [helpcontext(HlpCtxIRegister  + 12), helpstring("Read data from the memory
location")]
     HRESULT MoveIn16(
        [in] short space,
        [in] long offset,
        [in] long length,
        [out, retval] SAFEARRAY(short) *pBuf16);
     [helpcontext(HlpCtxIRegister  + 13), helpstring("Read data from the memory
location")]
     HRESULT MoveIn32(
        [in] short space,
        [in] long offset,
        [in] long length,
        [out, retval] SAFEARRAY(long) *pBuf32);
     [helpcontext(HlpCtxIRegister  + 14), helpstring("Write data to the memory
location")]
     HRESULT MoveOut8(
        [in] short space,
        [in] long offset,
        [in] long length,
```

```
        [in] SAFEARRAY(BYTE) *buf8);
     [helpcontext(HlpCtxIRegister  + 15), helpstring("Write data to the memory
location")]
     HRESULT MoveOut16(
        [in] short space,
        [in] long offset,
        [in] long length,
        [in] SAFEARRAY(short) *buf16);
     [helpcontext(HlpCtxIRegister  + 16), helpstring("Write data to the memory
location")]
     HRESULT MoveOut32(
        [in] short space,
        [in] long offset,
        [in] long length,
        [in] SAFEARRAY(long) *buf32);
     [helpcontext(HlpCtxIRegister  + 17), helpstring("Move data between memory
locations")]
     HRESULT Move(
        [in] short srcSpace,
        [in] long srcOffset,
        [in] DataWidth srcWidth,
        [in] short destSpace,
        [in] long destOffset,
        [in] DataWidth destWidth,
        [in] long length);
   };
```

Below is a table showing the methods of the IRegister interface and their equivalents in the VISA API.

IRegister Method	VISA Function
In8	viIn8
In16	viIn16
In32	viIn32
Out8	viOut8
Out16	viOut16
Out32	viOut32
MoveIn8	viMoveIn8
MoveIn16	viMoveIn16
MoveIn32	viMoveIn32
MoveOut8	viMoveOut8
MoveOut16	viMoveOut16
MoveOut32	viMoveOut32
Move	viMove

Table 5.1.3

Below is a table showing the COM properties of the IRegister interface and their corresponding VISA attributes.

IRegister Property	VISA Attribute
DestinationBigEndian	VI_ATTR_DEST_BYTE_ORDER
SourceBigEndian	VI_ATTR_SRC_BYTE_ORDER
DestinationIncrement	VI_ATTR_DEST_INCREMENT

SourceIncrement	VI_ATTR_SRC_INCREMENT

Table 5.1.4

RULE 5.1.25

Unless otherwise specified, all the methods and properties of IRegister **SHALL** behave identically to their VISA equivalents as defined in VPP 4.3.

RULE 5.1.26

The MoveX methods **SHALL** use SAFEARRAYs of the appropriate types instead of C arrays to transmit their data.

OBSERVATION 5.1.5

None of the low-level memory mapped methods and attributes are translated to VISA COM I/O. When a VISA COM I/O resource and the client communicating with it reside on different systems and DCOM is in use, low-level memory mapped regions cannot be dereferenced directly, and due to round-trip costs in DCOM, it is preferable to use the MoveX methods rather than the PeekX and PokeX methods of VISA C.

5.1.5. IRegister64 Interface

The IRegister64 interface augments the IRegister interface with functions that allow access to 64-bit integers and functions that allow 64-bit offsets. These functions mirror the 64-bit register functions added to VISA.

The IRegister64 interface is obsolete and has been replaced by the IRegister64_2 interface.

```
   [
      object,
      oleautomation,
            helpstring("Register Based Interface supporting 64-bit integers
(obsolete)"),
      uuid(DB8CBF29-D6D3-11D4-AA51-00A024EE30BD),
      helpcontext(HlpCtxIRegister64 + 49),
      pointer_default(unique),
      hidden
   ]
   interface IRegister64 : IRegister
   {
      [helpcontext(HlpCtxIRegister64  + 1), helpstring("Read a 64-bit integer
value from the memory location")]
      HRESULT In64(
         [in] short space,
         [in] long offset,
         [out, retval] __int64 *pVal8);

      [helpcontext(HlpCtxIRegister64  + 2), helpstring("Write a 64-bit integer
value to the memory location")]
      HRESULT Out64(
         [in] short space,
         [in] long offset,
         [in] __int64 val8);

      [helpcontext(HlpCtxIRegister64  + 3), helpstring("Read 64-bit integer data
from the memory location")]
      HRESULT MoveIn64(
         [in] short space,
         [in] long offset,
         [in] long length,
         [out, retval] SAFEARRAY(__int64) *pBuf8);

      [helpcontext(HlpCtxIRegister64  + 4), helpstring("Write 64-bit integer data
to the memory location")]
      HRESULT MoveOut64(
         [in] short space,
         [in] long offset,
         [in] long length,
         [in] SAFEARRAY(__int64) *buf8);
```

```
     [helpcontext(HlpCtxIRegister64  + 5), helpstring("Read a value from the
memory location")]
     HRESULT In8Ex(
        [in] short space,
        [in] __int64 offset,
        [out, retval] BYTE *pVal8);

     [helpcontext(HlpCtxIRegister64  + 6), helpstring("Read a value from the
memory location")]
     HRESULT In16Ex(
        [in] short space,
        [in] __int64 offset,
        [out, retval] short *pVal16);

     [helpcontext(HlpCtxIRegister64  + 7), helpstring("Read a value from the
memory location")]
     HRESULT In32Ex(
        [in] short space,
        [in] __int64 offset,
        [out, retval] long *pVal32);

     [helpcontext(HlpCtxIRegister64  + 8), helpstring("Read a value from the
memory location")]
     HRESULT In64Ex(
        [in] short space,
        [in] __int64 offset,
        [out, retval] __int64 *pVal8);

     [helpcontext(HlpCtxIRegister64  + 9), helpstring("Write a value to the
memory location")]
     HRESULT Out8Ex(
        [in] short space,
        [in] __int64 offset,
        [in] BYTE val8);

     [helpcontext(HlpCtxIRegister64  + 10), helpstring("Write a value to the
memory location")]
     HRESULT Out16Ex(
        [in] short space,
        [in] __int64 offset,
        [in] short val16);

     [helpcontext(HlpCtxIRegister64  + 11), helpstring("Write a value to the
memory location")]
     HRESULT Out32Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long val32);

     [helpcontext(HlpCtxIRegister64  + 12), helpstring("Write a value to the
memory location")]
     HRESULT Out64Ex(
        [in] short space,
        [in] __int64 offset,
        [in] __int64 val8);

     [helpcontext(HlpCtxIRegister64  + 13), helpstring("Read data from the
memory location")]
     HRESULT MoveIn8Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [out, retval] SAFEARRAY(BYTE) *pBuf8);

     [helpcontext(HlpCtxIRegister64  + 14), helpstring("Read data from the
memory location")]
     HRESULT MoveIn16Ex(
```

```
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [out, retval] SAFEARRAY(short) *pBuf16);

     [helpcontext(HlpCtxIRegister64  + 15), helpstring("Read data from the
memory location")]
     HRESULT MoveIn32Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [out, retval] SAFEARRAY(long) *pBuf32);

     [helpcontext(HlpCtxIRegister64  + 16), helpstring("Read data from the
memory location")]
     HRESULT MoveIn64Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [out, retval] SAFEARRAY(__int64) *pBuf8);

     [helpcontext(HlpCtxIRegister64  + 17), helpstring("Write data to the memory
location")]
     HRESULT MoveOut8Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [in] SAFEARRAY(BYTE) *buf8);

     [helpcontext(HlpCtxIRegister64  + 18), helpstring("Write data to the memory
location")]
     HRESULT MoveOut16Ex(
        [in] short space,
        [in] long offset,
        [in] __int64 length,
        [in] SAFEARRAY(short) *buf16);

     [helpcontext(HlpCtxIRegister64  + 19), helpstring("Write data to the memory
location")]
     HRESULT MoveOut32Ex(
        [in] short space,
        [in] long offset,
        [in] __int64 length,
        [in] SAFEARRAY(long) *buf32);

     [helpcontext(HlpCtxIRegister64  + 20), helpstring("Write data to the memory
location")]
     HRESULT MoveOut64Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [in] SAFEARRAY(__int64) *buf8);

     [helpcontext(HlpCtxIRegister64  + 21), helpstring("Move data between memory
locations")]
     HRESULT MoveEx(
        [in] short srcSpace,
        [in] __int64 srcOffset,
        [in] DataWidth srcWidth,
        [in] short destSpace,
        [in] __int64 destOffset,
        [in] DataWidth destWidth,
        [in] long length);
   };
```

IRegister64_2 Interface

The IRegister64_2 replaces the IRegister64 interface, and redefines the `MoveOut16Ex` and `MoveOut32Ex` methods to have the correct signature. Changes are highlighted below.

```
   [
      object,
      oleautomation,
      helpstring("Register Based Interface 2 supporting 64-bit integers"),
      uuid(DB8CBF2A-D6D3-11D4-AA51-00A024EE30BD),
      helpcontext(HlpCtxIRegister64 + 50),
      pointer_default(unique)
   ]
   interface IRegister64_2 : IRegister
   {
      [helpcontext(HlpCtxIRegister64  + 1), helpstring("Read a 64-bit integer
value from the memory location")]
      HRESULT In64(
         [in] short space,
         [in] long offset,
         [out, retval] __int64 *pVal8);

      [helpcontext(HlpCtxIRegister64  + 2), helpstring("Write a 64-bit integer
value to the memory location")]
      HRESULT Out64(
         [in] short space,
         [in] long offset,
         [in] __int64 val8);

      [helpcontext(HlpCtxIRegister64  + 3), helpstring("Read 64-bit integer data
from the memory location")]
      HRESULT MoveIn64(
         [in] short space,
         [in] long offset,
         [in] long length,
         [out, retval] SAFEARRAY(__int64) *pBuf8);

      [helpcontext(HlpCtxIRegister64  + 4), helpstring("Write 64-bit integer data
to the memory location")]
      HRESULT MoveOut64(
         [in] short space,
         [in] long offset,
         [in] long length,
         [in] SAFEARRAY(__int64) *buf8);

      [helpcontext(HlpCtxIRegister64  + 5), helpstring("Read a value from the
memory location")]
      HRESULT In8Ex(
         [in] short space,
         [in] __int64 offset,
         [out, retval] BYTE *pVal8);

      [helpcontext(HlpCtxIRegister64  + 6), helpstring("Read a value from the
memory location")]
      HRESULT In16Ex(
         [in] short space,
         [in] __int64 offset,
         [out, retval] short *pVal16);

      [helpcontext(HlpCtxIRegister64  + 7), helpstring("Read a value from the
memory location")]
      HRESULT In32Ex(
         [in] short space,
         [in] __int64 offset,
         [out, retval] long *pVal32);

      [helpcontext(HlpCtxIRegister64  + 8), helpstring("Read a value from the
```

```
memory location")]
     HRESULT In64Ex(
        [in] short space,
        [in] __int64 offset,
        [out, retval] __int64 *pVal8);

     [helpcontext(HlpCtxIRegister64  + 9), helpstring("Write a value to the
memory location")]
     HRESULT Out8Ex(
        [in] short space,
        [in] __int64 offset,
        [in] BYTE val8);

     [helpcontext(HlpCtxIRegister64  + 10), helpstring("Write a value to the
memory location")]
     HRESULT Out16Ex(
        [in] short space,
        [in] __int64 offset,
        [in] short val16);

     [helpcontext(HlpCtxIRegister64  + 11), helpstring("Write a value to the
memory location")]
     HRESULT Out32Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long val32);

     [helpcontext(HlpCtxIRegister64  + 12), helpstring("Write a value to the
memory location")]
     HRESULT Out64Ex(
        [in] short space,
        [in] int64 offset,
        [in] __int64 val8);

     [helpcontext(HlpCtxIRegister64  + 13), helpstring("Read data from the
memory location")]
     HRESULT MoveIn8Ex(
        [in] short space,
        [in] int64 offset,
        [in] long length,
        [out, retval] SAFEARRAY(BYTE) *pBuf8);

     [helpcontext(HlpCtxIRegister64  + 14), helpstring("Read data from the
memory location")]
     HRESULT MoveIn16Ex(
        [in] short space,
        [in] int64 offset,
        [in] long length,
        [out, retval] SAFEARRAY(short) *pBuf16);

     [helpcontext(HlpCtxIRegister64  + 15), helpstring("Read data from the
memory location")]
     HRESULT MoveIn32Ex(
        [in] short space,
        [in] int64 offset,
        [in] long length,
        [out, retval] SAFEARRAY(long) *pBuf32);

     [helpcontext(HlpCtxIRegister64  + 16), helpstring("Read data from the
memory location")]
     HRESULT MoveIn64Ex(
        [in] short space,
        [in] int64 offset,
        [in] long length,
        [out, retval] SAFEARRAY(__int64) *pBuf8);

     [helpcontext(HlpCtxIRegister64  + 17), helpstring("Write data to the memory
location")]
```

```
      HRESULT MoveOut8Ex(
         [in] short space,
         [in] int64 offset,
         [in] long length,
         [in] SAFEARRAY(BYTE) *buf8);

      [helpcontext(HlpCtxIRegister64  + 22), helpstring("Write data to the memory
location")]
      HRESULT MoveOut16Ex(
         [in] short space,
         [in] int64 offset,
         [in] long length,
         [in] SAFEARRAY(short) *buf16);

      [helpcontext(HlpCtxIRegister64  + 23), helpstring("Write data to the memory
location")]
      HRESULT MoveOut32Ex(
         [in] short space,
         [in] int64 offset,
         [in] long length,
         [in] SAFEARRAY(long) *buf32);

      [helpcontext(HlpCtxIRegister64  + 20), helpstring("Write data to the memory
location")]
      HRESULT MoveOut64Ex(
         [in] short space,
         [in] __int64 offset,
         [in] long length,
         [in] SAFEARRAY(__int64) *buf8);

      [helpcontext(HlpCtxIRegister64  + 21), helpstring("Move data between memory
locations")]
      HRESULT MoveEx(
         [in] short srcSpace,
         [in] __int64 srcOffset,
         [in] DataWidth srcWidth,
         [in] short destSpace,
         [in] __int64 destOffset,
         [in] DataWidth destWidth,
         [in] long length);
   };
```

Below is a table showing the methods of the IRegister64 and IRegister64_2 interfaces and their equivalents in the VISA API.

IRegister64/IRegister64_2 Method	VISA Function
In64	viIn64
Out64	viOut64
MoveIn64	viMoveIn64
MoveOut64	viMoveOut64
In8Ex	viIn8Ex
In16Ex	viIn16Ex
In32Ex	viIn32Ex
In64Ex	viIn64Ex
Out8Ex	viOut8Ex
Out16Ex	viOut16Ex
Out32Ex	viOut32Ex
Out64Ex	viOut64Ex

MoveIn8Ex	viMoveIn8Ex
MoveIn16Ex	viMoveIn16Ex
MoveIn32Ex	viMoveIn32Ex
MoveIn64Ex	viMoveIn64Ex
MoveOut8Ex	viMoveOut8Ex
MoveOut16Ex	viMoveOut16Ex
MoveOut32Ex	viMoveOut32Ex
MoveOut64Ex	viMoveOut64Ex
MoveEx	viMoveEx

Table 5.1.5

RULE 5.1.27

Unless otherwise specified, all the methods and properties of IRegister64 **SHALL** behave identically to their VISA equivalents as defined in VPP 4.3.

RULE 5.1.28

The MoveX methods **SHALL** use SAFEARRAYs of the appropriate types instead of C arrays to transmit their data.

OBSERVATION 5.1.6

None of the low-level memory mapped methods and attributes are translated to VISA COM I/O. When a VISA COM I/O resource and the client communicating with it reside on different systems and DCOM is in use, low-level memory mapped regions cannot be dereferenced directly, and due to round-trip costs in DCOM, it is preferable to use the MoveX methods rather than the PeekX and PokeX methods of VISA C.

RECOMMENDATION 5.1.3

All implementations of VISA-COM that have provided an implementation of IRegister64 should continue to implement IRegister64 for backwards compatibility.

5.1.6. ISharedRegister Interface

The ISharedRegister Interface provides a means of allocating memory on remote buses on INSTR sessions on interface types such as VXI. Below is the IDL specification for ISharedRegister.

```
[
   object,
   oleautomation,
   helpstring("Shared Memory Interface"),
   uuid(db8cbf08-d6d3-11d4-aa51-00a024ee30bd),
   helpcontext(HlpCtxISharedRegister + 49),
   pointer_default(unique)
]
interface ISharedRegister : IVisaSession
{
   [helpcontext(HlpCtxISharedRegister  + 1), helpstring("Allocate memory")]
   HRESULT AllocateMemory(
      [in] long size,
      [out, retval] long *pOffset);
   [helpcontext(HlpCtxISharedRegister  + 2), helpstring("Free memory")]
   HRESULT FreeMemory(
       [in] long offset);
};
```

Below is a table showing the methods of ISharedRegister and their VISA equivalents.

ISharedRegister Method	VISA Function

AllocateMemory	viMemAlloc
FreeMemory	viMemFree

Table 5.1.6

RULE 5.1.29

The methods of ISharedRegister **SHALL** behave identically to their equivalent VISA methods, as defined in VPP 4.3 unless noted otherwise in this document.

5.1.7. ISharedRegister64 Interface

The ISharedRegister64 Interface provides a means of allocating memory on remote buses on INSTR sessions on interface types such as VXI. Below is the IDL specification for ISharedRegister64.

```
   [
      object,
      oleautomation,
      helpstring("Shared Memory Interface supporting 64-bit integers"),
      uuid(DB8CBF26-D6D3-11D4-AA51-00A024EE30BD),
      helpcontext(HlpCtxISharedRegister64 + 49),
      pointer_default(unique)
   ]
   interface ISharedRegister64 : ISharedRegister
   {
      [helpcontext(HlpCtxISharedRegister64 + 1), helpstring("Allocate memory")]
      HRESULT AllocateMemoryEx(
         [in] long size,
         [out, retval] __int64 *pOffset);

      [helpcontext(HlpCtxISharedRegister64 + 2), helpstring("Free memory")]
      HRESULT FreeMemoryEx(
          [in] __int64 offset);
   };
```

Below is a table showing the methods of ISharedRegister and their VISA equivalents.

ISharedRegister64 Method	VISA Function
AllocateMemoryEx	viMemAllocEx
FreeMemoryEx	viMemFreeEx

Table 5.1.7

5.1.8. IGpib Interface

The IGpib Interface provides the INSTR attributes and methods specific to GPIB and GPIB-VXI INSTR sessions. Below is the IDL specification for IGpib.

```
   [
      object,
      oleautomation,
      helpstring("GPIB Interface"),
      uuid(db8cbf09-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIGpib + 49),
      pointer_default(unique)
   ]
   interface IGpib : IVisaSession
   {
      [propget, helpcontext(HlpCtxIGpib  + 1), helpstring("Get the primary
address")]
      HRESULT PrimaryAddress([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIGpib  + 2), helpstring("Get the REN line
state")]
```

```
     HRESULT RENState([out, retval] LineState *pVal);
     [propget, helpcontext(HlpCtxIGpib  + 3), helpstring("Get/Set whether to
repeat address")]
     HRESULT RepeatAddressingEnabled([out, retval] VARIANT_BOOL *pVal);
     [propput, helpcontext(HlpCtxIGpib  + 3), helpstring("Get/Set whether to
repeat address")]
     HRESULT RepeatAddressingEnabled([in] VARIANT_BOOL newVal);
     [propget, helpcontext(HlpCtxIGpib  + 4), helpstring("Get the secondary
address")]
     HRESULT SecondaryAddress([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIGpib  + 5), helpstring("Get/Set whether to
unaddress")]
     HRESULT UnaddressingEnabled([out, retval] VARIANT_BOOL *pVal);
     [propput, helpcontext(HlpCtxIGpib  + 5), helpstring("Get/Set whether to
unaddress")]
     HRESULT UnaddressingEnabled([in] VARIANT_BOOL newVal);

     [helpcontext(HlpCtxIGpib  + 6), helpstring("Control the REN line
(remote/local) state")]
     HRESULT ControlREN(
         [in] RENControlConst mode);
  };
```

The following table lists all the methods of IGpib and their equivalents in VISA.

IGpib Method	VISA Function
ControlREN	viGpibControlREN

Table 5.1.8

The following table lists all the COM properties of IGpib and their equivalents in VISA.

IGpib Property	VISA Attribute
PrimaryAddress	VI_ATTR_GPIB_PRIMARY_ADDR
RENState	VI_ATTR_GPIB_REN_STATE
RepeatAddressingEnabled	VI_ATTR_GPIB_READDR_EN
SecondaryAddress	VI_ATTR_GPIB_SECONDARY_ADDR
UnaddressingEnabled	VI_ATTR_GPIB_UNADDR_EN

Table 5.1.9

RULE 5.1.30

All the methods and properties in IGpib **SHALL** have the same behavior as their VISA equivalents, as defined in VPP 4.3 unless otherwise noted in this document.

5.1.9. ISerial Interface

The ISerial interface provides the methods and properties specific to ASRL INSTR sessions. Below is the IDL specification of the ISerial Interface.

```
   [
      object,
      oleautomation,
      helpstring("Serial Interface"),
      uuid(db8cbf0c-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxISerial + 49),
      pointer_default(unique)
   ]
   interface ISerial : IVisaSession
   {
      [propget, helpcontext(HlpCtxISerial  + 1), helpstring("Get the number of
bytes available")]
      HRESULT BytesAvailable([out, retval] long *pVal);
      [propget, helpcontext(HlpCtxISerial  + 2), helpstring("Get/Set the baud
rate")]
      HRESULT BaudRate([out, retval] long *pVal);
      [propput, helpcontext(HlpCtxISerial  + 2), helpstring("Get/Set the baud
rate")]
      HRESULT BaudRate([in] long newVal);
      [propget, helpcontext(HlpCtxISerial  + 3), helpstring("Get/Set the number
of data bits")]
      HRESULT DataBits([out, retval] short *pVal);
      [propput, helpcontext(HlpCtxISerial  + 3), helpstring("Get/Set the number
of data bits")]
      HRESULT DataBits([in] short newVal);
      [propget, helpcontext(HlpCtxISerial  + 4), helpstring("Get the CTS line
state")]
      HRESULT ClearToSendState([out, retval] LineState *pVal);
      [propget, helpcontext(HlpCtxISerial  + 5), helpstring("Get the DCD line
state")]
      HRESULT DataCarrierDetectState([out, retval] LineState *pVal);
      [propget, helpcontext(HlpCtxISerial  + 6), helpstring("Get the DSR line
state")]
      HRESULT DataSetReadyState([out, retval] LineState *pVal);
      [propget, helpcontext(HlpCtxISerial  + 7), helpstring("Get/Set the DTR line
state")]
      HRESULT DataTerminalReadyState([out, retval] LineState *pVal);
      [propput, helpcontext(HlpCtxISerial  + 7), helpstring("Get/Set the DTR line
state")]
      HRESULT DataTerminalReadyState([in] LineState newVal);
      [propget, helpcontext(HlpCtxISerial  + 8), helpstring("Get/Set the input
end mode")]
      HRESULT EndIn([out, retval] SerialEndConst *pVal);
```

```
     [propput, helpcontext(HlpCtxISerial  + 8), helpstring("Get/Set the input
end mode")]
     HRESULT EndIn([in] SerialEndConst newVal);
     [propget, helpcontext(HlpCtxISerial  + 9), helpstring("Get/Set the output
end mode")]
     HRESULT EndOut([out, retval] SerialEndConst *pVal);
     [propput, helpcontext(HlpCtxISerial  + 9), helpstring("Get/Set the output
end mode")]
     HRESULT EndOut([in] SerialEndConst newVal);
     [propget, helpcontext(HlpCtxISerial  + 10), helpstring("Get/Set the flow
control")]
     HRESULT FlowControl([out, retval] SerialFlowControl *pVal);
     [propput, helpcontext(HlpCtxISerial  + 10), helpstring("Get/Set the flow
control")]
     HRESULT FlowControl([in] SerialFlowControl newVal);
     [propget, helpcontext(HlpCtxISerial  + 11), helpstring("Get/Set the
parity")]
     HRESULT Parity([out, retval] SerialParity *pVal);
     [propput, helpcontext(HlpCtxISerial  + 11), helpstring("Get/Set the
parity")]
     HRESULT Parity([in] SerialParity newVal);
     [propget, helpcontext(HlpCtxISerial  + 12), helpstring("Get the RI line
state")]
     HRESULT RingIndicatorState([out, retval] LineState *pVal);
     [propget, helpcontext(HlpCtxISerial  + 13), helpstring("Get/Set the RTS
line state")]
     HRESULT RequestToSendState([out, retval] LineState *pVal);
     [propput, helpcontext(HlpCtxISerial  + 13), helpstring("Get/Set the RTS
line state")]
     HRESULT RequestToSendState([in] LineState newVal);
     [propget, helpcontext(HlpCtxISerial  + 14), helpstring("Get/Set the number
of stop bits")]
     HRESULT StopBits([out, retval] SerialStopBits *pVal);
     [propput, helpcontext(HlpCtxISerial  + 14), helpstring("Get/Set the number
of stop bits")]
     HRESULT StopBits([in] SerialStopBits newVal);
     [propget, helpcontext(HlpCtxISerial  + 15), helpstring("Get/Set the error
replacement character")]
     HRESULT ReplacementCharacter([out, retval] BYTE *pVal);
     [propput, helpcontext(HlpCtxISerial  + 15), helpstring("Get/Set the error
replacement character")]
     HRESULT ReplacementCharacter([in] BYTE newVal);
     [propget, helpcontext(HlpCtxISerial  + 16), helpstring("Get/Set the XON
character")]
     HRESULT XONCharacter([out, retval] BYTE *pVal);
     [propput, helpcontext(HlpCtxISerial  + 16), helpstring("Get/Set the XON
character")]
     HRESULT XONCharacter([in] BYTE newVal);
     [propget, helpcontext(HlpCtxISerial  + 17), helpstring("Get/Set the XOFF
character")]
     HRESULT XOFFCharacter([out, retval] BYTE *pVal);
     [propput, helpcontext(HlpCtxISerial  + 17), helpstring("Get/Set the XOFF
character")]
     HRESULT XOFFCharacter([in] BYTE newVal);

     [helpcontext(HlpCtxISerial  + 18), helpstring("Set the serial receive or
transmit buffer size")]
     HRESULT SetBufferSize(
        [in] BufferMask mask,
        [in] long size);
     [helpcontext(HlpCtxISerial  + 19), helpstring("Flush the specified serial
buffer")]
     HRESULT Flush(
        [in, defaultvalue(IO_IN_OUT_BUF)] BufferMask mask,
        [in, defaultvalue(FALSE)] VARIANT_BOOL discard);
   };
```

The following table lists all the ISerial methods and their equivalent VISA functions.

ISerial Method	VISA Function
SetBufferSize	viSetBuf
Flush	viFlush

Table 5.1.10

The following table lists all the ISerial COM properties and their equivalent VISA attributes.

ISerial Property	VISA Attribute
BytesAvailable	VI_ATTR_ASRL_AVAIL_NUM
BaudRate	VI_ATTR_ASRL_BAUD
DataBits	VI_ATTR_ASRL_DATA_BITS
ClearToSendState	VI_ATTR_ASRL_CTS_STATE
DataCarrierDetectState	VI_ATTR_ASRL_DCD_STATE
DataSetReadyState	VI_ATTR_ASRL_DSR_STATE
DataTerminalReadyState	VI_ATTR_ASRL_DTR_STATE
EndIn	VI_ATTR_ASRL_END_IN
EndOut	VI_ATTR_ASRL_END_OUT
FlowControl	VI_ATTR_ASRL_FLOW_CNTRL
Parity	VI_ATTR_ASRL_PARITY
RingIndicatorState	VI_ATTR_ASRL_RI_STATE
RequestToSendState	VI_ATTR_ASRL_RTS_STATE
StopBits	VI_ATTR_ASRL_STOP_BITS
ReplacementCharacter	VI_ATTR_ASRL_REPLACE_CHAR
XONCharacter	VI_ATTR_ASRL_XON_CHAR
XOFFCharacter	VI_ATTR_ASRL_XOFF_CHAR

Table 5.1.11

RULE 5.1.31

The methods and properties of the ISerial interface **SHALL** behave identically to their VISA equivalents as defined by VPP 4.3 unless otherwise noted.

RULE 5.1.32

The methods Flush and SetBufferSize **SHALL** only allow changes to RS-232 settings rather than the more general behavior of the viFlush and viSetBuf methods. **IF** the mask parameter is for a buffer other than the RS-232 buffer, **THEN** these methods **SHALL** return an HRESULT of E_INVALIDARG.

5.1.10. IVxi Interface

The IVxi interface defines the methods and COM properties specific to VXI and GPIB-VXI INSTR session VISA COM I/O components. Below is the IDL specification for IVxi.

```
   [
      object,
      oleautomation,
      helpstring("VXI Interface (obsolete)"),
      uuid(db8cbf0f-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIVxi + 49),
      pointer_default(unique),
      hidden
   ]
   interface IVxi : IVisaSession
   {
      [propget, helpcontext(HlpCtxIVxi  + 1), helpstring("Get the commander's
logical address")]
      HRESULT CommanderLA([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIVxi  + 2), helpstring("Get/Set the target
address modifier")]
      HRESULT DestinationAccessPrivilege([out, retval] VXIMemoryAccessPrivilege
*pVal);
      [propput, helpcontext(HlpCtxIVxi  + 2), helpstring("Get/Set the target
address modifier")]
      HRESULT DestinationAccessPrivilege([in] VXIMemoryAccessPrivilege newVal);
      [propget, helpcontext(HlpCtxIVxi  + 3), helpstring("Get the VXI device
class")]
      HRESULT DeviceClass([out, retval] VXIDevClass *pVal);
      [propget, helpcontext(HlpCtxIVxi  + 4), helpstring("Get/Set the FDC channel
number")]
      HRESULT FastDataChannel([out, retval] short *pVal);
      [propput, helpcontext(HlpCtxIVxi  + 4), helpstring("Get/Set the FDC channel
number")]
      HRESULT FastDataChannel([in] short newVal);
      [propget, helpcontext(HlpCtxIVxi  + 5), helpstring("Get/Set the FDC mode")]
      HRESULT FastDataChannelMode([out, retval] FDCMode *pVal);
      [propput, helpcontext(HlpCtxIVxi  + 5), helpstring("Get/Set the FDC mode")]
      HRESULT FastDataChannelMode([in] FDCMode newVal);
      [propget, helpcontext(HlpCtxIVxi  + 6), helpstring("Get/Set whether to use
an FDC channel pair")]
      HRESULT FastDataChannelUsePair([out, retval] VARIANT_BOOL *pVal);
      [propput, helpcontext(HlpCtxIVxi  + 6), helpstring("Get/Set whether to use
an FDC channel pair")]
      HRESULT FastDataChannelUsePair([in] VARIANT_BOOL newVal);
      [propget, helpcontext(HlpCtxIVxi  + 7), helpstring("Get whether the device
is this controller's servant")]
      HRESULT ImmediateServant([out, retval] VARIANT_BOOL *pVal);
      [propget, helpcontext(HlpCtxIVxi  + 8), helpstring("Get the logical
address")]
      HRESULT LogicalAddress([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIVxi  + 9), helpstring("Get the mainframe's
logical address")]
      HRESULT MainframeLogicalAddress([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIVxi  + 10), helpstring("Get the manufacturer
ID")]
      HRESULT ManufacturerID([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIVxi  + 11), helpstring("Get the manufacturer
name")]
      HRESULT ManufacturerName([out, retval] BSTR *pVal);
      [propget, helpcontext(HlpCtxIVxi  + 12), helpstring("Get the memory base
address")]
      HRESULT MemoryBase([out, retval] long *pVal);
      [propget, helpcontext(HlpCtxIVxi  + 13), helpstring("Get the memory size")]
      HRESULT MemorySize([out, retval] long *pVal);
      [propget, helpcontext(HlpCtxIVxi  + 14), helpstring("Get the memory
space")]
```

```
     HRESULT MemorySpace([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIVxi  + 15), helpstring("Get the model code")]
     HRESULT ModelCode([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIVxi  + 16), helpstring("Get the model name")]
     HRESULT ModelName([out, retval] BSTR *pVal);
     [propget, helpcontext(HlpCtxIVxi  + 17), helpstring("Get/Set the trigger
ID")]
     HRESULT TriggerID([out, retval] TriggerLine *pVal);
     [propput, helpcontext(HlpCtxIVxi  + 17), helpstring("Get/Set the trigger
ID")]
     HRESULT TriggerID([in] TriggerLine newVal);
     [propget, helpcontext(HlpCtxIVxi  + 18), helpstring("Get the device's
slot")]
     HRESULT Slot([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIVxi  + 19), helpstring("Get/Set the source
address modifier")]
     HRESULT SourceAccessPrivilege([out, retval] VXIMemoryAccessPrivilege
*pVal);
     [propput, helpcontext(HlpCtxIVxi  + 19), helpstring("Get/Set the source
address modifier")]
     HRESULT SourceAccessPrivilege([in] VXIMemoryAccessPrivilege newVal);
     [propget, helpcontext(HlpCtxIVxi  + 20), helpstring("Get which trigger
lines are supported")]
     HRESULT TriggerSupport([out, retval] long *pVal);

     [helpcontext(HlpCtxIVxi  + 21), helpstring("Assert a trigger")]
     HRESULT AssertTrigger(
         [in, defaultvalue(TRIG_PROT_DEFAULT)] TriggerProtocol protocol);
     [helpcontext(HlpCtxIVxi  + 22), helpstring("Send a miscellaneous VXI
command or query")]
     HRESULT CommandQuery(
        [in] VXICommandQuery mode,
        [in] long cmd,
        [out, retval] long *pResponse);
  };
```

The following table lists all the IVxi methods and their equivalent VISA functions.

IVxi Method	VISA Function
CommandQuery	viVxiCommandQuery

Table 5.1.12

The following table lists all the IVxi COM properties and their equivalent VISA attributes.

IVxi Property	VISA Attribute
CommanderLA	VI_ATTR_CMDR_LA
DestinationAccessPriviledge	VI_ATTR_DEST_ACCESS_PRIV
DeviceClass	VI_ATTR_VXI_DEV_CLASS
FastDataChannel	VI_ATTR_FDC_CHNL
FastDataChannelMode	VI_ATTR_FDC_MODE
FastDataChannelUsePair	VI_ATTR_FDC_USE_PAIR
ImmediateServant	VI_ATTR_IMMEDIATE_SERV
Logical Address	VI_ATTR_VXI_LA
MainframeLogicalAddress	VI_ATTR_MAINFRAME_LA
ManufacturerID	VI_ATTR_MANF_ID
ManufacturerName	VI_ATTR_MANF_NAME

MemoryBase	VI_ATTR_MEM_BASE
MemorySize	VI_ATTR_MEM_SIZE
MemorySpace	VI_ATTR_MEM_SPACE
ModelCode	VI_ATTR_MODEL_CODE
ModelName	VI_ATTR_MODEL_NAME
Slot	VI_ATTR_SLOT
SourceAccessPrivilege	VI_ATTR_SRC_ACCESS_PRIV
TriggerSupport	VI_ATTR_VXI_TRIG_SUPPORT
TriggerID	VI_ATTR_TRIG_ID

Table 5.1.13

RULE 5.1.33

The methods and properties of the IVxi interface **SHALL** behave identically to their VISA equivalents as defined in VPP 4.3 unless otherwise noted.

5.1.11. IVxi3 Interface

```
[
     object,
     oleautomation,
     helpstring("VXI Interface"),
     uuid(db8cbf22-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxIVxi3 + 49),
     pointer_default(unique)
]
interface IVxi3 : IVxi
{
     [propget, helpcontext(HlpCtxIVxi3  + 1), helpstring("Get 488.2
Compliance")]
     HRESULT Is4882Compliant([out, retval] VARIANT_BOOL *pVal);
};
```

The following table lists all the IVxi3 COM properties and their equivalent VISA attributes.

IVxi3 Property	VISA Attribute
Is4882Compliant	VI_ATTR_4882_COMPLIANT

Table 5.1.14

RULE 5.1.34

The property of the IVxi3 interface **SHALL** behave identically to the its VISA equivalent as defined in VPP 4.3 unless otherwise noted.

5.1.12. ITcpipInstr Interface

```
   [
      object,
      oleautomation,
      helpstring("TCP/IP Instrument Interface"),
      uuid(db8cbf0d-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxITcpipInstr + 49),
      pointer_default(unique)
   ]
   interface ITcpipInstr : IVisaSession
   {
      [propget, helpcontext(HlpCtxITcpipInstr  + 1), helpstring("Get the TCP/IP
address")]
```

```
     HRESULT Address([out, retval] BSTR *pVal);
     [propget, helpcontext(HlpCtxITcpipInstr  + 2), helpstring("Get the TCP/IP
hostname")]
     HRESULT HostName([out, retval] BSTR *pVal);
     [propget, helpcontext(HlpCtxITcpipInstr  + 3), helpstring("Get the LAN
device name")]
     HRESULT DeviceName([out, retval] BSTR *pVal);
   };
```

The following table lists all the ITcpipInstr COM properties and their equivalent VISA attributes.

ITcpipInstr Property	VISA Attribute
Address	VI_ATTR_TCPIP_ADDR
HostName	VI_ATTR_TCPIP_HOSTNAME
DeviceName	VI_ATTR_TCPIP_DEVICE_NAME

Table 5.1.15

5.1.13. IUsb Interface

```
[
     object,
     oleautomation,
     helpstring("USB Interface"),
     uuid(db8cbf24-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxIUsb + 49),
     pointer_default(unique)
]
interface IUsb : IVisaSession
{
     [propget, helpcontext(HlpCtxIUsb  + 1), helpstring("Get the manufacturer
 ID")]
     HRESULT ManufacturerID([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIUsb  + 2), helpstring("Get the manufacturer
 name")]
     HRESULT ManufacturerName([out, retval] BSTR *pVal);
     [propget, helpcontext(HlpCtxIUsb  + 3), helpstring("Get the model code")]
     HRESULT ModelCode([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIUsb  + 4), helpstring("Get the model name")]
     HRESULT ModelName([out, retval] BSTR *pVal);
     [propget, helpcontext(HlpCtxIUsb  + 5), helpstring("Get 488.2 Compliance")]
     HRESULT Is4882Compliant([out, retval] VARIANT_BOOL *pVal);
     [propget, helpcontext(HlpCtxIUsb  + 6), helpstring("Get the USB Serial
 Number")]
     HRESULT UsbSerialNumber([out, retval] BSTR *pVal);
     [propget, helpcontext(HlpCtxIUsb  + 7), helpstring("Get the USB Interface
 Number")]
     HRESULT UsbInterfaceNumber([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIUsb  + 8), helpstring("Get the USB Protocol")]
     HRESULT UsbProtocol([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIUsb  + 9), helpstring("Get/Set the Maximum
 Interrupt Size")]
     HRESULT MaximumInterruptSize([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIUsb  + 9), helpstring("Get/Set the Maximum
 Interrupt Size")]
     HRESULT MaximumInterruptSize([in] short size);

     [helpcontext(HlpCtxIUsb  + 10), helpstring("Control the REN line
 (remote/local) state")]
     HRESULT ControlREN(
        [in] RENControlConst mode);
     [helpcontext(HlpCtxIUsb  + 11), helpstring("Send Data to the USB Control
 Port")]
```

```
    HRESULT ControlOut(
       [in] short bmRequestType,
       [in] short bRequest,
       [in] short wValue,
       [in] short wIndex,
       [in] short wLength,
       [in] SAFEARRAY(BYTE) *buffer);
    [helpcontext(HlpCtxIUsb  + 12), helpstring("Request Data from the USB
Control Port")]
    HRESULT ControlIn(
       [in] short bmRequestType,
       [in] short bRequest,
       [in] short wValue,
       [in] short wIndex,
       [in] short wLength,
       [out, retval] SAFEARRAY(BYTE) *pBuf);
};
```

The following table lists all the IUsb COM properties and their equivalent VISA attributes.

IUsb Property	VISA Attribute
ManufacturerID	VI_ATTR_MANF_ID
ManufacturerName	VI_ATTR_MANF_NAME
ModelCode	VI_ATTR_MODEL_CODE
ModelName	VI_ATTR_MODEL_NAME
Is4882Compliant	VI_ATTR_4882_COMPLIANT
UsbSerialNumber	VI_ATTR_USB_SERIAL_NUM
UsbInterfaceNumber	VI_ATTR_USB_INTFC_NUM
MaximumInterruptSize	VI_ATTR_USB_MAX_INTR_SIZE
UsbProtocol	VI_ATTR_USB_PROTOCOL

Table 5.1.16

The following table lists all the IUsb COM methods and their equivalent VISA functions.

IUsb Method	VISA Function
ControlREN	viGpibControlREN
ControlOut	viUsbControlOut
ControlIn	viUsbControlIn

Table 5.1.17

RULE 5.1.35

The methods and properties of the IUsb interface **SHALL** behave identically to their VISA equivalents as defined in VPP 4.3 unless otherwise noted.

RULE 5.1.36

VISA COM I/O resources **SHALL** implement these methods as specified in VPP 4.3 except where specified otherwise in this specification.

5.1.14. IHislipInstr Interface

```
[
   object,
   oleautomation,
   helpstring("High Speed LAN Protocol (HiSLIP) Instrument Interface"),
```

```
     uuid(DB8CBF27-D6D3-11D4-AA51-00A024EE30BD),
     helpcontext(HlpCtxIHislipInstr + 49),
     pointer_default(unique)
   ]
   interface IHislipInstr : ITcpipInstr
   {
     [propget, helpcontext(HlpCtxIHislipInstr  + 1), helpstring("Get the
negotiated HiSLIP protocol version")]
     HRESULT ProtocolVersion([out, retval] long *pVal);

     [propget, helpcontext(HlpCtxIHislipInstr  + 2), helpstring("Get/Set the
HiSLIP Maximum Message Size in KB (1024 bytes)")]
     HRESULT MaxMessage([out, retval] long *pVal);
     [propput, helpcontext(HlpCtxIHislipInstr  + 2), helpstring("Get/Set the
HiSLIP Maximum Message Size in KB (1024 bytes)")]
     HRESULT MaxMessage([in] long newVal);

     [propget, helpcontext(HlpCtxIHislipInstr  + 3), helpstring("Get/Set the
HiSLIP Operlap Enabled")]
     HRESULT OverlapEnabled([out, retval] VARIANT_BOOL *pVal);
     [propput, helpcontext(HlpCtxIHislipInstr  + 3), helpstring("Get/Set the
HiSLIP Overlap Enabled ")]
     HRESULT OverlapEnabled([in] VARIANT_BOOL newVal);

     [helpcontext(HlpCtxIHislipInstr  + 4), helpstring("Control the REN line
(remote/local) state")]
     HRESULT ControlREN(
         [in] RENControlConst mode);
};
```

The following table lists all the IHislipInstr COM properties and their equivalent VISA attributes.

IHislipInstr Property	VISA Attribute
ProtocolVersion	VI_ATTR_TCPIP_HISLIP_VERSION
MaxMessage	VI_ATTR_TCPIP_HISLIP_MAX_MESSAGE_KB
OverlapEnabled	VI_ATTR_TCPIP_HISLIP_OVERLAP_EN

Table 5.1.18

5.1.15. IPxi Interface

```
   [
     object,
     oleautomation,
     helpstring("PXI Interface"),
     uuid(DB8CBF28-D6D3-11D4-AA51-00A024EE30BD),
     helpcontext(HlpCtxIPxi + 49),
     pointer_default(unique)
   ]
   interface IPxi : IVisaSession
   {
     [propget, helpcontext(HlpCtxIPxi  + 1), helpstring("Get the PCI bus
number")]
     HRESULT BusNumber([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 2), helpstring("Get the PCI device
number")]
     HRESULT DevNumber([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 3), helpstring("Get the PCI function
number")]
     HRESULT FuncNumber([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 4), helpstring("Get the slot path")]
     HRESULT SlotPath([out, retval] BSTR *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 5), helpstring("Get the slot number or
```

```
special feature connected to local left bus lines")]
     HRESULT SlotLocalBusLeft([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 6), helpstring("Get the slot number or
special feature connected to local right bus lines")]
     HRESULT SlotLocalBusRight([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 7), helpstring("Get the trigger bus
number of this device")]
     HRESULT TriggerBus([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 8), helpstring("Get the PXI star
trigger bus")]
     HRESULT StarTriggerBus([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 9), helpstring("Get the connected PXI
star line")]
     HRESULT StarTriggerLine([out, retval] short *pVal);

     [propget, helpcontext(HlpCtxIPxi  + 10), helpstring("Get the memory type
used in BAR 0")]
     HRESULT MemTypeBar0([out, retval] PXIMemType *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 11), helpstring("Get the memory type
used in BAR 1")]
     HRESULT MemTypeBar1([out, retval] PXIMemType *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 12), helpstring("Get the memory type
used in BAR 2")]
     HRESULT MemTypeBar2([out, retval] PXIMemType *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 13), helpstring("Get the memory type
used in BAR 3")]
     HRESULT MemTypeBar3([out, retval] PXIMemType *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 14), helpstring("Get the memory type
used in BAR 4")]
     HRESULT MemTypeBar4([out, retval] PXIMemType *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 15), helpstring("Get the memory type
used in BAR 5")]
     HRESULT MemTypeBar5([out, retval] PXIMemType *pVal);

     [propget, helpcontext(HlpCtxIPxi  + 16), helpstring("Get the memory base
address for BAR 0")]
     HRESULT MemBaseBar0([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 17), helpstring("Get the memory base
address for BAR 1")]
     HRESULT MemBaseBar1([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 18), helpstring("Get the memory base
address for BAR 2")]
     HRESULT MemBaseBar2([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 19), helpstring("Get the memory base
address for BAR 3")]
     HRESULT MemBaseBar3([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 20), helpstring("Get the memory base
address for BAR 4")]
     HRESULT MemBaseBar4([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 21), helpstring("Get the memory base
address for BAR 5")]
     HRESULT MemBaseBar5([out, retval] long *pVal);

     [propget, helpcontext(HlpCtxIPxi  + 22), helpstring("Get the memory size
for BAR 0")]
     HRESULT MemSizeBar0([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 23), helpstring("Get the memory size
for BAR 1")]
     HRESULT MemSizeBar1([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 24), helpstring("Get the memory size
for BAR 2")]
     HRESULT MemSizeBar2([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 25), helpstring("Get the memory size
for BAR 3")]
     HRESULT MemSizeBar3([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 26), helpstring("Get the memory size
for BAR 4")]
     HRESULT MemSizeBar4([out, retval] long *pVal);
```

```
     [propget, helpcontext(HlpCtxIPxi  + 27), helpstring("Get the memory size
for BAR 5")]
     HRESULT MemSizeBar5([out, retval] long *pVal);

     [propget, helpcontext(HlpCtxIPxi  + 28), helpstring("Get the chassis
number")]
     HRESULT ChassisNumber([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 29), helpstring("Get whether the device
is PXI Express")]
     HRESULT IsExpress([out, retval] VARIANT_BOOL *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 30), helpstring("Get the link width
used by the slot")]
     HRESULT SlotLinkWidth([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 31), helpstring("Get the maximum usable
link width")]
     HRESULT MaxLinkWidth([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 32), helpstring("Get the negotiated
link width")]
     HRESULT ActualLinkWidth([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 33), helpstring("Get the differential
star bus number")]
     HRESULT DstarBusNumber([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 34), helpstring("Get the connected set
of PXI Express differential star bus lines")]
     HRESULT DstarLineSet([out, retval] short *pVal);
   };
```

The following table lists all the IPxi COM properties and their equivalent VISA attributes.

IPxi Property	VISA Attribute
BusNumber	VI_ATTR_PXI_BUS_NUM
DevNumber	VI_ATTR_PXI_DEV_NUM
FuncNumber	VI_ATTR_PXI_FUNC_NUM
SlotPath	VI_ATTR_PXI_SLOTPATH
SlotLocalBusLeft	VI_ATTR_PXI_SLOT_LBUS_LEFT
SlotLocalBusRight	VI_ATTR_PXI_SLOT_LBUS_RIGHT
TriggerBus	VI_ATTR_PXI_TRIG_BUS
StarTriggerBus	VI_ATTR_PXI_STAR_TRIG_BUS
StarTriggerLine	VI_ATTR_PXI_STAR_TRIG_LINE
MemTypeBar0	VI_ATTR_PXI_MEM_TYPE_BAR0
MemTypeBar1	VI_ATTR_PXI_MEM_TYPE_BAR1
MemTypeBar2	VI_ATTR_PXI_MEM_TYPE_BAR2
MemTypeBar3	VI_ATTR_PXI_MEM_TYPE_BAR3
MemTypeBar4	VI_ATTR_PXI_MEM_TYPE_BAR4
MemTypeBar5	VI_ATTR_PXI_MEM_TYPE_BAR5
MemBaseBar0	VI_ATTR_PXI_MEM_BASE_BAR0
MemBaseBar1	VI_ATTR_PXI_MEM_BASE_BAR1
MemBaseBar2	VI_ATTR_PXI_MEM_BASE_BAR2
MemBaseBar3	VI_ATTR_PXI_MEM_BASE_BAR3
MemBaseBar4	VI_ATTR_PXI_MEM_BASE_BAR4
MemBaseBar5	VI_ATTR_PXI_MEM_BASE_BAR5

MemSizeBar0	VI_ATTR_PXI_MEM_SIZE_BAR0
MemSizeBar1	VI_ATTR_PXI_MEM_SIZE_BAR1
MemSizeBar2	VI_ATTR_PXI_MEM_SIZE_BAR2
MemSizeBar3	VI_ATTR_PXI_MEM_SIZE_BAR3
MemSizeBar4	VI_ATTR_PXI_MEM_SIZE_BAR4
MemSizeBar5	VI_ATTR_PXI_MEM_SIZE_BAR5
ChassisNumber	VI_ATTR_PXI_CHASSIS
IsExpress	VI_ATTR_PXI_IS_EXPRESS
SlotLinkWidth	VI_ATTR_PXI_SLOT_LWIDTH
MaxLinkWidth	VI_ATTR_PXI_MAX_LWIDTH
ActualLinkWidth	VI_ATTR_PXI_ACTUAL_LWIDTH
DstarBusNumber	VI_ATTR_PXI_DSTAR_BUS
DstarLineSet	VI_ATTR_PXI_DSTAR_SET

Table 5.1.19

The following table lists all the IPxi COM methods and their equivalent VISA functions.

IPxi Method	VISA Function
AssertTrigger	viAssertTrigger

Table 5.1.20

5.2. MEMACC Resources

Memory Access (MEMACC) VISA COM I/O resources encapsulate the address space of a memory mapped bus such as the VXIbus. The MEMACC VISA COM I/O resources provide many of the same interfaces as INSTR resources that provide register access. It should be noted here that no address mapping or window services are provided in VISA COM I/O because of limitations of the COM calling conventions necessary to provide remote method invocation functionality.

RULE 5.2.1
All VISA COM I/O MEMACC resources **SHALL** implement the IRegister and IVxiMemacc Interfaces.

RULE 5.2.2
All VISA COM I/O GPIB-VXI MEMACC resources **SHALL** implement the IGpib interface defined in Section 5.1, *Instrument Control Resource*.

RULE 5.2.3
VISA COM I/O resources **SHALL** return E_NOINTERFACE when QueryInterface'd for an interface defined by VISA COM I/O other than the ones explicitly required or allowed to be implemented.

5.2.1 IVxiMemacc Interface

The IVxiMemacc Interface provides the properties specific to VXI Memory Access resources. These properties are a subset of the IVxi interface. Below is the IDL specification for the IVxiMemacc interface.

```
   [
      object,
      oleautomation,
      helpstring("VXI Memory Access Interface"),
      uuid(db8cbf10-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIVxiMemacc + 49),
      pointer_default(unique)
   ]
   interface IVxiMemacc : IRegister
   {
      [propget, helpcontext(HlpCtxIVxiMemacc  + 1), helpstring("Get/Set the
target address modifier")]
      HRESULT DestinationAccessPrivilege([out, retval] VXIMemoryAccessPrivilege
*pVal);
      [propput, helpcontext(HlpCtxIVxiMemacc  + 1), helpstring("Get/Set the
target address modifier")]
      HRESULT DestinationAccessPrivilege([in] VXIMemoryAccessPrivilege newVal);
      [propget, helpcontext(HlpCtxIVxiMemacc  + 2), helpstring("Get/Set the
source address modifier")]
      HRESULT SourceAccessPrivilege([out, retval] VXIMemoryAccessPrivilege
*pVal);
      [propput, helpcontext(HlpCtxIVxiMemacc  + 2), helpstring("Get/Set the
source address modifier")]
      HRESULT SourceAccessPrivilege([in] VXIMemoryAccessPrivilege newVal);
      [propget, helpcontext(HlpCtxIVxiMemacc  + 3), helpstring("Get the logical
address")]
      HRESULT LogicalAddress([out, retval] short *pVal);
   };
```

The following table lists all the IVxiMemacc COM properties and their equivalent VISA attributes.

IVxiMemacc Property	VISA Attribute
DestinationAccessPrivilege	VI_ATTR_DEST_ACCESS_PRIV
SourceAccessPriviledge	VI_ATTR_SRC_ACCESS_PRIV
LogicalAddress	VI_ATTR_VXI_LA

Table 5.2.1

RULE 5.2.4

The properties of the IVxiMemacc interface **SHALL** behave identically to their equivalent VISA attributes, as defined in VPP 4.3 unless noted otherwise in this document.

5.3. INTFC Resources

The only INTFC VISA COM I/O resource, GPIB INTFC, provides an interface-level view of the GPIB bus and provides properties and methods to interact with the GPIB interface.

RULE 5.3.1
GPIB INTFC VISA COM I/O resources **SHALL** implement the interfaces IGpibIntfc and IGpibIntfcMessage.

RULE 5.3.2
VISA COM I/O resources **SHALL** return E_NOINTERFACE when QueryInterface'd for an interface defined by VISA COM I/O other than the ones explicitly required or allowed to be implemented.

5.3.1. IGpibIntfc Interface

The IGpibIntfc interface provides the properties and methods specific to GPIB INTFC sessions, except for messaging capabilities. These properties and methods tend to be an extension to the properties and methods present in the IGpib interface.

```
   [
      object,
      oleautomation,
      helpstring("Board-level GPIB Interface"),
      uuid(db8cbf0a-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIGpibIntfc + 49),
      pointer_default(unique)
   ]
   interface IGpibIntfc : IVisaSession
   {
      [propget, helpcontext(HlpCtxIGpibIntfc  + 1), helpstring("Get the
controller addressing state")]
      HRESULT AddressingState([out, retval] GPIBAddressState *pVal);
      [propget, helpcontext(HlpCtxIGpibIntfc  + 2), helpstring("Get the ATN line
state")]
      HRESULT ATNState([out, retval] LineState *pVal);
      [propget, helpcontext(HlpCtxIGpibIntfc  + 3), helpstring("Get/Set the
status byte")]
      HRESULT DevStatusByte([out, retval] BYTE *pVal);
      [propput, helpcontext(HlpCtxIGpibIntfc  + 3), helpstring("Get/Set the
status byte")]
      HRESULT DevStatusByte([in] BYTE newVal);
      [propget, helpcontext(HlpCtxIGpibIntfc  + 4), helpstring("Get the
controller CIC state")]
      HRESULT CICState([out, retval] VARIANT_BOOL *pVal);
      [propget, helpcontext(HlpCtxIGpibIntfc  + 5), helpstring("Get/Set the HS-
488 cable length")]
      HRESULT HS488CBLLength([out, retval] short *pVal);
      [propput, helpcontext(HlpCtxIGpibIntfc  + 5), helpstring("Get/Set the HS-
488 cable length")]
      HRESULT HS488CBLLength([in] short newVal);
      [propget, helpcontext(HlpCtxIGpibIntfc  + 6), helpstring("Get the NDAC
line state")]
      HRESULT NDACState([out, retval] LineState *pVal);
      [propget, helpcontext(HlpCtxIGpibIntfc  + 7), helpstring("Get/Set the
primary address")]
      HRESULT PrimaryAddress([out, retval] short *pVal);
      [propput, helpcontext(HlpCtxIGpibIntfc  + 7), helpstring("Get/Set the
primary address")]
      HRESULT PrimaryAddress([in] short newVal);
      [propget, helpcontext(HlpCtxIGpibIntfc  + 8), helpstring("Get the REN line
state")]
      HRESULT RENState([out, retval] LineState *pVal);
      [propget, helpcontext(HlpCtxIGpibIntfc  + 9), helpstring("Get/Set the
secondary address")]
      HRESULT SecondaryAddress([out, retval] short *pVal);
      [propput, helpcontext(HlpCtxIGpibIntfc  + 9), helpstring("Get/Set the
```

```
secondary address")]
       HRESULT SecondaryAddress([in] short newVal);
       [propget, helpcontext(HlpCtxIGpibIntfc  + 10), helpstring("Get the SRQ
line state")]
       HRESULT SRQState([out, retval] LineState *pVal);
       [propget, helpcontext(HlpCtxIGpibIntfc  + 11), helpstring("Get/Set the
system controller state")]
       HRESULT SysControlState([out, retval] VARIANT_BOOL *pVal);
       [propput, helpcontext(HlpCtxIGpibIntfc  + 11), helpstring("Get/Set the
system controller state")]
       HRESULT SysControlState([in] VARIANT_BOOL newVal);

       [helpcontext(HlpCtxIGpibIntfc  + 12), helpstring("Write GPIB command bytes
on the bus")]
       HRESULT Command(
              [in] SAFEARRAY(BYTE) *buffer,
              [in] long count,
              [out, retval] long *pRetCount);
       [helpcontext(HlpCtxIGpibIntfc  + 13), helpstring("Control the ATN line
state")]
       HRESULT ControlATN(
               [in] ATNControlConst mode);
       [helpcontext(HlpCtxIGpibIntfc  + 14), helpstring("Control the REN line
(remote/local) state")]
       HRESULT ControlREN(
               [in] RENControlConst mode);
       [helpcontext(HlpCtxIGpibIntfc  + 15), helpstring("Pass control to the
specified device")]
       HRESULT PassControl(
              [in] short primAddr,
              [in, defaultvalue(-1)] short secAddr);
       [helpcontext(HlpCtxIGpibIntfc  + 16), helpstring("Pulse the IFC line")]
       HRESULT SendIFC();
   };
```

The following table lists all the IGpibIntfc COM properties and their equivalent VISA attributes.

IGpibIntfc Property	VISA Attribute
AddressingState	VI_ATTR_GPIB_ADDR_STATE
ATNState	VI_ATTR_GPIB_ATN_STATE
CICState	VI_ATTR_GPIB_CIC_STATE
DevStatusByte	VI_ATTR_DEV_STATUS_BYTE
HS488CBLLength	VI_ATTR_GPIB_HS488_CBL_LEN
NDACState	VI_ATTR_GPIB_NDAC_STATE
PrimaryAddress	VI_ATTR_GPIB_PRIMARY_ADDR
RENState	VI_ATTR_GPIB_REN_STATE
SecondaryAddress	VI_ATTR_GPIB_SECONDARY_ADDR
SRQState	VI_ATTR_GPIB_SRQ_STATE
SysControlState	VI_ATTR_GPIB_SYS_CNTRL_STATE

Table 5.3.1

The following table lists all the IGpibIntfc COM methods and their equivalent VISA functions.

IGpibIntfc Method	VISA Function
Command	viGpibCommand

ControlATN	viGpibControlATN
ControlREN	viGpibControlREN
PassControl	viGpibPassControl
SendIFC	viGpibSendIFC

Table 5.3.2

RULE 5.3.3

The methods and properties of the IGpibIntfc interface **SHALL** behave identically to their VISA equivalents unless otherwise noted in this document.

5.3.2. IGpibIntfcMessage Interface

The IGpibIntfcMessage interface provides the subset of text stream features present on a GPIB INTFC resource.

```
   [
      object,
      oleautomation,
      helpstring("Board-level GPIB Message Based Interface"),
      uuid(db8cbf0b-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIGpibIntfcMessage + 49),
      pointer_default(unique)
   ]
   interface IGpibIntfcMessage : IVisaSession
   {
      [propget, helpcontext(HlpCtxIGpibIntfcMessage  + 1), helpstring("Get/Set
whether to assert END on Write")]
      HRESULT SendEndEnabled([out, retval] VARIANT_BOOL *pVal);
      [propput, helpcontext(HlpCtxIGpibIntfcMessage  + 1), helpstring("Get/Set
whether to assert END on Write")]
      HRESULT SendEndEnabled([in] VARIANT_BOOL newVal);
      [propget, helpcontext(HlpCtxIGpibIntfcMessage  + 2), helpstring("Get/Set
the termination character")]
      HRESULT TerminationCharacter([out, retval] BYTE *pVal);
      [propput, helpcontext(HlpCtxIGpibIntfcMessage  + 2), helpstring("Get/Set
the termination character")]
      HRESULT TerminationCharacter([in] BYTE newVal);
      [propget, helpcontext(HlpCtxIGpibIntfcMessage  + 3), helpstring("Get/Set
whether to use the termination character on Read")]
      HRESULT TerminationCharacterEnabled([out, retval] VARIANT_BOOL *pVal);
      [propput, helpcontext(HlpCtxIGpibIntfcMessage  + 3), helpstring("Get/Set
whether to use the termination character on Read")]
      HRESULT TerminationCharacterEnabled([in] VARIANT_BOOL newVal);

      [helpcontext(HlpCtxIGpibIntfcMessage  + 4), helpstring("Assert a trigger")]
      HRESULT AssertTrigger(
          [in, defaultvalue(TRIG_PROT_DEFAULT)] TriggerProtocol protocol);
      [helpcontext(HlpCtxIGpibIntfcMessage  + 5), helpstring("Read the specified
number of bytes")]
      HRESULT Read(
         [in] long count,
         [out, retval] SAFEARRAY(BYTE) *pBuffer);
      [helpcontext(HlpCtxIGpibIntfcMessage  + 6), helpstring("Read the specified
number of bytes as a string")]
      HRESULT ReadString(
         [in] long count,
         [out, retval] BSTR *pBuffer);
      [helpcontext(HlpCtxIGpibIntfcMessage  + 7), helpstring("Write the specified
data")]
      HRESULT Write(
         [in] SAFEARRAY(BYTE) *buffer,
         [in] long count,
         [out, retval] long *pRetCount);
      [helpcontext(HlpCtxIGpibIntfcMessage  + 8), helpstring("Write the specified
```

```
string")]
     HRESULT WriteString(
        [in] BSTR buffer,
        [out, retval] long *pRetCount);
  };
```

The following table lists all the IGpibIntfcMessage COM properties and their equivalent VISA attributes.

IGpibIntfcMessage Property	VISA Attribute
SendEndEnabled	VI_ATTR_SEND_END_EN
TerminationCharacter	VI_ATTR_TERMCHAR
TerminationCharacterEnabled	VI_ATTR_TERMCHAR_EN

Table 5.3.3

The following table lists all the IGpibIntfcMessage COM methods and their equivalent VISA functions.

IGpibIntfcMessage Method	VISA Function
AssertTrigger	viAssertTrigger
Read	viRead
ReadString	viRead
Write	viWrite
WriteString	viWrite

Table 5.3.4

RULE 5.3.4
Unless otherwise noted, the methods and properties of IGpibIntfcMessage **SHALL** behave identically to their equivalents in VISA.

RULE 5.3.5
Both Read and Write **SHALL** use SAFEARRAYs of unsigned characters to retrieve and send stream data.

RULE 5.3.6
The Write method **SHALL** return the HRESULT E_INVALIDARG or the equivalent VISA HRESULT if the parameter count is larger than the size of the SAFEARRAY passed in.

RECOMMENDATION 5.3.1
It is recommended that upon an invalid count parameter, there should be an IErrorInfo structure placed on the thread-local storage that describes the error more specifically.

RULE 5.3.7
If the Write method is called with the parameter count smaller than the size of the SAFEARRAY passed in, only the first count bytes **SHALL** be written to the instrument resource.

RULE 5.3.8
The status parameter **SHALL** equal the return value used by viRead and viWrite in VISA upon the return of the methods Read and Write.

OBSERVATION 5.3.1
Although COM APIs, like C APIs can return errors as the return value of functions/methods, many COM environments have problems understanding or ignore return values that are successful other than S_OK, so successful return values that indicate various success conditions are not feasible in COM.

RULE 5.3.9
The ReadString and WriteString methods **SHALL** behave identically to the Read and Write methods but will give and receive BSTRs instead of SAFEARRAYs of BYTEs.

RULE 5.3.10

WriteString SHALLfail with the error code E_VISA_INV_FMT when one or more of the Unicode characters in the Message argument have an ambiguous or no valid conversion to ASCII.

5.4. SOCKET Resources

The only SOCKET session type defined for VISA COM I/O resources is the TCPIP SOCKET resource. This resource provides low-level access to a TCPIP stream. SOCKET resources are close enough in behavior to INSTR resources that they can be used polymorphically with INSTR resources for messaging services, that is, they implement the basic messaging interfaces.

RULE 5.4.1

All VISA COM I/O TCPIP SOCKET resources **SHALL** implement the interfaces IBaseMessage, IMessage, IAsyncMessage, and ITcpipSocket.

RULE 5.4.2

VISA COM I/O resources **SHALL** return E_NOINTERFACE when QueryInterface'd for an interface defined by VISA COM I/O other than the ones explicitly required or allowed to be implemented.

5.4.1. ITcpipSocket Interface

The ITcpipSocket interface provides the VISA COM I/O properties and methods specific to TCPIP SOCKET resource sessions.

```
   [
      object,
      oleautomation,
      helpstring("TCP/IP Socket Interface"),
      uuid(db8cbf0e-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxITcpipSocket + 49),
      pointer_default(unique)
   ]
   interface ITcpipSocket : IVisaSession
   {
      [propget, helpcontext(HlpCtxITcpipSocket  + 1), helpstring("Get the TCP/IP
address")]
      HRESULT Address([out, retval] BSTR *pVal);
      [propget, helpcontext(HlpCtxITcpipSocket  + 2), helpstring("Get the TCP/IP
hostname")]
      HRESULT HostName([out, retval] BSTR *pVal);
      [propget, helpcontext(HlpCtxITcpipSocket  + 3), helpstring("Get/Set
whether to send keep-alive packets")]
      HRESULT KeepAlive([out, retval] VARIANT_BOOL *pVal);
      [propput, helpcontext(HlpCtxITcpipSocket  + 3), helpstring("Get/Set
whether to send keep-alive packets")]
      HRESULT KeepAlive([in] VARIANT_BOOL newVal);
      [propget, helpcontext(HlpCtxITcpipSocket  + 4), helpstring("Get/Set
whether to use the Nagle algorithm")]
      HRESULT NoDelay([out, retval] VARIANT_BOOL *pVal);
      [propput, helpcontext(HlpCtxITcpipSocket  + 4), helpstring("Get/Set
whether to use the Nagle algorithm")]
      HRESULT NoDelay([in] VARIANT_BOOL newVal);
      [propget, helpcontext(HlpCtxITcpipSocket  + 5), helpstring("Get the TCP/IP
port")]
      HRESULT Port([out, retval] short *pVal);

      [helpcontext(HlpCtxITcpipSocket  + 6), helpstring("Set the socket receive
or transmit buffer size")]
      HRESULT SetBufferSize(
         [in] BufferMask mask,
         [in] long size);
      [helpcontext(HlpCtxITcpipSocket  + 7), helpstring("Flush the specified
socket buffer")]
      HRESULT Flush(
         [in, defaultvalue(IO_IN_OUT_BUF)] BufferMask mask,
         [in, defaultvalue(FALSE)] VARIANT_BOOL discard);
   };
```

The following table lists all the ITcpipSocket COM properties and their equivalent VISA attributes.

ITcpipSocket Property	VISA Attribute
Address	VI_ATTR_TCPIP_ADDR
HostName	VI_ATTR_TCPIP_HOSTNAME
KeepAlive	VI_ATTR_TCPIP_KEEPALIVE
NoDelay	VI_ATTR_TCPIP_NODELAY
Port	VI_ATTR_TCPIP_PORT

Table 5.4.1

RULE 5.4.3

Unless otherwise noted, the properties of ITcpipSocket **SHALL** behave identically to their equivalents in VISA.

5.5. BACKPLANE Resources

Currently, the only BACKPLANE session type defined is the VXI BACKPLANE resource. The BACKPLANE resource lets a controller query and manipulate specific lines on a specific mainframe in a given VXI system. Services are provided to map, unmap, assert, and receive hardware triggers, and also to assert various utility and interrupt signals. This includes advanced functionality that may not be available in all implementations or all vendors' controllers. This resource differs from other resources in that they provide no communication (messaging or register) operations.

RULE 5.5.1
All VXI BACKPLANE VISA COM I/O resources **SHALL** implement the interface IVxiBackplane.

RULE 5.5.2
VXI BACKPLANE VISA COM I/O resources **SHALL** return E_NOINTERFACE when QueryInterface'd for an interface defined by VISA COM I/O other than the ones explicitly required or allowed to be implemented.

5.5.1. IVxiBackplane Interface

The IVxiBackplane interface provides the properties and methods specific to the VXI BACKPLANE resource.

```
   [
      object,
      oleautomation,
      helpstring("VXI Backplane Interface"),
      uuid(db8cbf11-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIVxiBackplane + 49),
      pointer_default(unique)
   ]
   interface IVxiBackplane : IVisaSession
   {
      [propget, helpcontext(HlpCtxIVxiBackplane  + 1), helpstring("Get the
mainframe's logical address")]
      HRESULT MainframeLA([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIVxiBackplane  + 2), helpstring("Get/Set the
trigger ID")]
      HRESULT TriggerId([out, retval] TriggerLine *pVal);
      [propput, helpcontext(HlpCtxIVxiBackplane  + 2), helpstring("Get/Set the
trigger ID")]
      HRESULT TriggerId([in] TriggerLine newVal);
      [propget, helpcontext(HlpCtxIVxiBackplane  + 3), helpstring("Get which
trigger lines are asserted")]
      HRESULT TriggerStatus([out, retval] long *pVal);
      [propget, helpcontext(HlpCtxIVxiBackplane  + 4), helpstring("Get which
trigger lines are supported")]
      HRESULT TriggerSupport([out, retval] long *pVal);
      [propget, helpcontext(HlpCtxIVxiBackplane  + 5), helpstring("Get which
interrupt lines are asserted")]
      HRESULT VxiVmeInterruptStatus([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIVxiBackplane  + 6), helpstring("Get the
SYSFAIL line state")]
      HRESULT VxiVmeSysfailStatus([out, retval] LineState *pVal);

      [helpcontext(HlpCtxIVxiBackplane  + 7), helpstring("Assert the specified
interrupt or signal")]
      HRESULT AssertInterruptSignal(
         [in] AssertInterruptConst mode,
         [in] long statusID);
      [helpcontext(HlpCtxIVxiBackplane  + 8), helpstring("Assert a trigger")]
      HRESULT AssertTrigger(
          [in, defaultvalue(TRIG_PROT_DEFAULT)] TriggerProtocol protocol);
      [helpcontext(HlpCtxIVxiBackplane  + 9), helpstring("Assert or deassert the
specified utility signal")]
      HRESULT AssertUtilSignal(
```

```
         [in] AssertUtilityConst line);
     [helpcontext(HlpCtxIVxiBackplane  + 10), helpstring("Map between the
specified trigger lines")]
     HRESULT MapTrigger(
        [in] TriggerLine trigSrc,
        [in] TriggerLine trigDest,
        [in, defaultvalue(0)] short mode);
     [helpcontext(HlpCtxIVxiBackplane  + 11), helpstring("Undo a previous
trigger line mapping")]
     HRESULT UnmapTrigger(
        [in] TriggerLine trigSrc,
        [in, defaultvalue(TRIG_ALL)] TriggerLine trigDest);
   };
```

The following table lists all the IVxiBackplane COM properties and their equivalent VISA attributes.

IVxiBackplane Property	VISA Attribute
MainframeLA	VI_ATTR_MAINFRAME_LA
TriggerID	VI_ATTR_TRIG_ID
TriggerStatus	VI_ATTR_VXI_TRIG_STATUS
TriggerSupport	VI_ATTR_VXI_TRIG_SUPPORT
VxiVmeInterruptStatus	VI_ATTR_VXI_VME_INTR_STATUS
VxiVmeSysfailStatus	VI_ATTR_VXI_VME_SYSFAIL_STATE

Table 5.5.1

The following table lists all the IVxiBackplane COM methods and their equivalent VISA functions.

IVxiBackplane Method	VISA Function
AssertInterruptSignal	viAssertIntrSignal
AssertTrigger	viAssertTrigger
AssertUtilSignal	viAssertUtilSignal
MapTrigger	viMapTrigger
UnmapTrigger	viUnmapTrigger

Table 5.5.2

RULE 5.5.3

VISA COM I/O resources **SHALL** implement these methods as specified in VPP 4.3 except where specified otherwise in this specification.

Section 6: VISA COM I/O Components and Installation

Section 2.6 described the components that are required for a complete VISA COM I/O implementation. This section covers the details of the installation and gives detailed requirements of the components' implementation.

The installation of the components includes registry entries that need to be adjusted or added and where to place files on the system hard drive.

The components of a VISA COM I/O implementation have several implementation requirements to ensure successful runtime interoperability. The specification builds on the Microsoft COM specification for VISA COM I/O Components.

6.1. Installation of VISA COM I/O Components

In order for users to reference and use the VISA COM I/O libraries, several requirements of COM and of the libraries have to be met: the COM system must be able to locate and use the VISA COM I/O type library, the VISA COM I/O system must be able to create an instance of the Global Resource Manager, the Global Resource Manager must be able to enumerate and Create the Vendor-Specific Resource Managers (SRMs), and the SRMs must be able to use COM to create the Resource Components they are designed to find and instantiate.

Another goal is for the VISA COM I/O shared components to remain on the users' systems until all VISA COM I/O implementations are removed and that the versions of the components on the system be identical to the latest version of the shared components redistributed in the install programs of the VISA COM I/O implementations installed on the system.

The following specifications for the installation of the Global Shared Components refer not only to the capabilities and behaviors of the components themselves, but also the requirements regarding install behavior for VISA COM I/O implementations and vendor components.

6.1.1. Global Resource Manager and Conflict Table Manager Components

The Global Resource Manager Component is a common component with a well-defined GUID so that it is creatable on any system that has VISA COM I/O installed on it. Below is a table of the registry entries that are set when the DLL entry point DllRegisterServer of the Global Resource Managers DLL is called and removed when DllUnregisterServer is called.

RULE 6.1.1

The DllRegisterServer entry point of the Global Resource Manager's DLL **SHALL** add the described keys to the registry and the DllUnregisterServer entry point **SHALL** remove them.

Registry Key Location	Value(s)
HKCR\CLSID\{db8cbf1c-d6d3-11d4-aa51-00a024ee30bd}	@="VISA COM I/O Global Resource Manager Class"
HKCR\CLSID\{db8cbf1c-d6d3-11d4-aa51-00a024ee30bd}\InprocServer32	@="($VXIPNPPATH)\VisaCom\GlobMgr.dll" ThreadingModel="Both"
HKCR\CLSID\{db8cbf1c-d6d3-11d4-aa51-00a024ee30bd}\ProgID	@="VISA.GlobalRM.1"
HKCR\CLSID\{db8cbf1c-d6d3-11d4-aa51-00a024ee30bd}\VersionIndependentProgID	@="VISA.GlobalRM"
HKCR\VISA.GlobalRM\	@="VISA COM I/O Global Resource Manager Class"
HKCR\VISA.GlobalRM\CLSID	@="{db8cbf1c-d6d3-11d4-aa51-00a024ee30bd}"
HKCR\VISA.GlobalRM\CurVer	@="VISA.GlobalRM.1"
HKCR\VISA.GlobalRM.1	@="VISA COM I/O Global Resource Manager Class"
HKCR\VISA.GlobalRM.1\CLSID	@="{db8cbf1c-d6d3-11d4-aa51-00a024ee30bd}"

RULE 6.1.2

The DllRegisterServer entry point of the Conflict Table Manager's DLL **SHALL** add the described keys to the registry and the DllUnregisterServer entry point **SHALL** remove them.

Registry Key Location	Value(s)
HKCR\CLSID\{db8cbf1f-d6d3-11d4-aa51-00a024ee30bd}	@="VISA COM I/O Resource Conflict Manager"
HKCR\CLSID\{db8cbf1f-d6d3-11d4-aa51-00a024ee30bd} \InprocServer32	@="($VXIPNPPATH)\VisaCom\GlobMgr.dll" ThreadingModel="Both"
HKCR\CLSID\{db8cbf1f-d6d3-11d4-aa51-00a024ee30bd} \ProgID	@="VISA.ConflictMgr.1"
HKCR\CLSID\{db8cbf1f-d6d3-11d4-aa51-00a024ee30bd} \VersionIndependentProgID	@="VISA.ConflictMgr"
HKCR\VISA.ConflictMgr\	@="VISA COM I/O Resource Conflict Manager"
HKCR\VISA.ConflictMgr\CLSID	@="{db8cbf1f-d6d3-11d4-aa51-00a024ee30bd}"
HKCR\VISA.ConflictMgr\CurVer	@="VISA.ConflictMgr.1"
HKCR\VISA.ConflictMgr.1	@="VISA COM I/O Resource Conflict Manager"
HKCR\VISA.ConflictMgr.1\CLSID	@="{db8cbf1f-d6d3-11d4-aa51-00a024ee30bd}"

RULE 6.1.3

The DllRegisterServer and DllUnregisterServer entry points of the Global Resource Manager's DLL **SHALL** use the appropriate Win32 APIs as defined by the COM specification to register and unregister the types in the VISA COM I/O type library.

6.1.2. Basic Formatted I/O Component

The Basic Formatted I/O Component has a well known GUID and ProgID (which are placed in the registry) so that users can write code that references the component and will work across VISA COM I/O implementations.

RULE 6.1.4

The DllRegisterServer entry point of the Basic Formatted I/O Component's DLL **SHALL** add the described keys to the registry and the DllUnregisterServer entry point **SHALL** remove them.

Registry Key Location	Value(s)
HKCR\CLSID\{db8cbf1d-d6d3-11d4-aa51-00a024ee30bd}	@="VISA COM I/O Basic Formatted I/O Class"
HKCR\CLSID\{db8cbf1d-d6d3-11d4-aa51-00a024ee30bd}\InprocServer32	@="($VXIPNPPATH)\VisaCom\BasFrmIO.dll" ThreadingModel="Both"
HKCR\CLSID\{db8cbf1d-d6d3-11d4-aa51-00a024ee30bd}\ProgID	@="VISA.BasicFormattedIO.1"
HKCR\CLSID\{db8cbf1d-d6d3-11d4-aa51-00a024ee30bd}\VersionIndependentProgID	@="VISA.BasicFormattedIO"
HKCR\VISA.BasicFormattedIO\	@="VISA COM I/O Basic Formatted I/O Class"
HKCR\VISA.BasicFormattedIO\CLSID	@="{db8cbf1d-d6d3-11d4-aa51-00a024ee30bd}"
HKCR\VISA.BasicFormattedIO\CurVer	@="VISA.BasicFormattedIO.1"
HKCR\VISA.BasicFormattedIO.1	@="VISA COM I/O Basic Formatted I/O Class"
HKCR\VISA.BasicFormattedIO.1\CLSID	@="{db8cbf1d-d6d3-11d4-aa51-00a024ee30bd}"

6.1.3. Vendor-Specific Resource Manager

The Vendor-Specific Resource Manager needs to register itself so that the Global Resource Manager can locate and instantiate (CoCreateEx) it. There is a standard registry-based method for locating COM components that provide a functionality-class called "Category ID". In addition to registering in this method, a second, performance-oriented method that is also registry-based **SHALL** be used.

RULE 6.1.5

The DllRegisterServer entry point of the Vendor Specific Resource Manager Component's DLL **SHALL** add the described keys to the registry and the DllUnregisterServer entry point **SHALL** remove them.

Registry Key Location	Value(s)
HKCR\CLSID\{<VENDOR-CHOSEN GUID>}	@="<Class Description>"
HKCR\CLSID\{<VENDOR-CHOSEN GUID>}\InprocServer32	@="<Full Path to Vendor-Chosen DLL Filename>" ThreadingModel="Both"
HKCR\CLSID\{<VENDOR-CHOSEN GUID>}\ProgID	@="<Vendor Name>.<Class Name>.<Version Number>"
HKCR\CLSID\{<VENDOR-CHOSEN GUID>}\VersionIndependentProgID	@="<Vendor Name>.<Class Name>"
HKCR\<Vendor Name>.<Class Name>\	@="<Class Description>"
HKCR\<Vendor Name>.<Class Name>\CLSID	@="{<Vendor-Chosen GUID>}"

HKCR\<Vendor Name>.<Class Name>\CurVer	@="<Vendor Name>.<Class Name>.<Version Number>"
HKCR\<Vendor Name>.<Class Name>.<Version Number>	@="<Class Description>"
HKCR\<Vendor Name>.<Class Name>.<Version Number>\CLSID	@="{<Vendor-Chosen GUID>}"
HKCR\VISA.GlobalRM \VendorSpecificRMs\<CLSID of SRM>	@="<Class Description>"

RULE 6.1.6

IF a Vendor Specific Resource Manager provides its own type library, **THEN** its DllRegisterServer and DllUnregisterServer entry points of **SHALL** use the appropriate Win32 APIs as defined by the COM specification to register and unregister the types in the Vendor's type library.

RULE 6.1.7

The Vendor-Specific Resource Manager **SHALL NOT** register or unregister any of the types declared in the VISA COM I/O type library, and **SHALL NOT** register or unregister the VISA COM I/O type library in its DLL entry points.

RULE 6.1.8

The DllRegisterServer and DllUnregisterServer entry points of the Vendor-Specific Resource Manager Component's DLL **SHALL** CoCreate the Component Category Manager and use the ICatRegister interface to Register Category ID {db8cbf21-d6d3-11d4-aa51-00a024ee30bd} with the Locale ID 0x0409 and the description "VISA COM I/O Vendor-Specific Resource Manager Classes". The entry points **SHALL** register and unregister the CLSID of the SRM as implementing this Category ID using the interface.

OBSERVATION 6.1.1

While the locations of the registry entries for CatIDs are well known, it is better to rely on Microsoft's Component Category Manager to do the registration to guarantee future support.

6.1.4. VISA COM I/O Resource Component

The VISA COM I/O Resource Component needs to register itself so that the Global Resource Manager can locate and instantiate (CoCreateEx) it. There is a standard registry-based method for locating COM components that provide a functionality-class called "Category ID". In addition to registering in this method, a second, performance-oriented method that is also registry-based **SHALL** be used.

RULE 6.1.9

The DllRegisterServer entry point of a VISA COM I/O Resource Component's DLL **SHALL** add the described keys to the registry and the DllUnregisterServer entry point **SHALL** remove them.

Registry Key Location	Value(s)
HKCR\CLSID\{<VENDOR-CHOSEN GUID>}	@="<Class Description>"
HKCR\CLSID\{<VENDOR-CHOSEN GUID>}\InprocServer32	@="<Full Path to Vendor-Chosen DLL Filename>" ThreadingModel="Both"
HKCR\CLSID\{<VENDOR-CHOSEN GUID>}\ProgID	@="<Vendor Name>.<Class Name>.<Version Number>"
HKCR\CLSID\{<VENDOR-CHOSEN GUID>}\VersionIndependentProgID	@="<Vendor Name>.<Class Name>"
HKCR\<Vendor Name>.<Class Name>\	@="<Class Description>"
HKCR\<Vendor Name>.<Class Name>\CLSID	@="{<Vendor-Chosen GUID>}"
HKCR\<Vendor Name>.<Class Name>\CurVer	@="<Vendor Name>.<Class Name>.<Version Number>"
HKCR\<Vendor Name>.<Class Name>.<Version Number>	@="<Class Description>"
HKCR\<Vendor Name>.<Class Name>.<Version Number>\CLSID	@="{<Vendor-Chosen GUID>}"

RULE 6.1.10

A VISA COM I/O Resource Component **SHALL NOT** register or unregister any of the types declared in the VISA COM I/O type library, AND **SHALL NOT** register or unregister the VISA COM I/O type library in its DLL entry points.

6.1.5. General Installation Requirements for Vendor Specific Components

RULE 6.1.11

Each VISA COM I/O implementation **SHALL** provide one Vendor-Specific Resource Manager Component and one or more Resource Components.

RULE 6.1.12

A VISA COM I/O implementation's SRM **SHALL** be able to find and/or open the exact set of all the Resource Components belonging to the implementation.

RULE 6.1.13

Each VISA COM I/O Resource Component **SHALL** be able to locate and access all resources associated with the physical resource the Resource Component communicates with.

RECOMMENDATION 6.1.1

VISA COM I/O implementations should provide a help file that contains at least the error descriptions pointed to by the IErrorInfo structures returned with the vendor's resources' errors.

RECOMMENDATION 6.1.2

VISA COM I/O vendors should install their components in a subdirectory of either "{$VXIPNPPATH}\{VENDORNAME}" or "{$PROGRAMFILES}\Common Files\{VENDORNAME}" to avoid filename conflicts with other vendors' components.

OBSERVATION 6.1.2

Unlike VPP-4.3.2 and VPP-4.3.3, which rely on a single file named visa32.dll, a VISA COM I/O implementation has no name requirements. This allows both COM-based and non-COM-based implementations to reside side-by-side on the same system. Since the full pathname to each COM component is in the registry, the installation path requirements are also more flexible.

RULE 6.1.14

A vendor's VISA uninstaller or its SRM uninstaller SHALL NOT silently uninstall the VISA COM Standard Components.

On Windows Vista, Windows 7, and Windows 8, if a vendor's VISA installer calls the VISA COM Standard Components installer, it **SHALL** invoke the VISA COM Standard Components installer with admin privileges.

6.2. Implementation of VISA COM I/O Components

Each of the components described in this specification has the following implementation requirements.

6.2.1. Global Resource Manager

Runtime performance of the Global Resource Manager should be as fast as possible with as small a memory footprint as feasible.

The implementation of the Global Resource Manager's main interfaces, IResourceManager and IResourceManager3, is described in Section 4.3, *The Global Resource Manager Component*. There are a few additional requirements of the component.

RULE 6.2.1
The Global Resource Manager Component **SHALL** be thread-safe and runnable in single-threaded and multi-threaded apartments.

RULE 6.2.2
IF the Global Resource Manager version being installed is newer than the installed version on the target system, **THEN** the installer **SHALL** register the VISA COM Type Library that is built into GlobMgr.dll.

RECOMMENDATION 6.2.1
It is strongly recommended that the VISA COM Type Library not be installed as a separate TLB file. This will prevent version inconsistencies since the TLB file type is not versioned in a way that installers handle.

OBSERVATION 6.2.1

The only time a TLB file should be placed on any non-development system is between the time the interface is first created and the time that the version of GlobMgr.dll that contains that interface is sanctioned.

RULE 6.2.3
IF a vendor's installer or installed software ever changed the marshalling method of any VISA COM I/O defined interfaces, **THEN** that vendor's uninstaller **SHALL** re-register the VISA COM Type Library that is built into GlobMgr.dll.

RECOMMENDATION 6.2.2
It is strongly recommended that only the Universal marshaller be used for the VISA COM I/O defined interfaces.

RULE 6.2.4
Any vendor's COM component which marshals VISA COM-defined interfaces **SHALL** be installed with the Component Category ID "VISA COM I/O Custom Marshaller" with the GUID {db8cbf25-d6d3-11d4-aa51-00a024ee30bd}.

OBSERVATION 6.2.2
There are many possible side-effects of registering a marshaller other than the Universal Marshaller for the VISA COM interfaces, so to make problems caused by those installations more diagnosable and supportable, they need to be identifiable. This is a service that Component Category ID's provide.

PERMISSION 6.2.1
The Global Resource Manager **MAY** use the free-threaded marshaller for in-process marshalling.

OBSERVATION 6.2.3
Using the free-threaded marshaller allows the component to avoid the cost of marshalling between apartments in the same process but requires more work to be thread-safe.

6.2.2. Basic Formatted I/O Component

The implementation of the Formatted I/O component's main interface, IFormattedIO488, is described in Section 7, *Formatted I/O*. There are a few additional requirements pertaining to the component itself.

RULE 6.2.5

The Basic Formatted I/O Component **SHALL** be thread-safe and runnable in single-threaded and multi-threaded apartments.

RULE 6.2.6

IF a vendor's installer or installed software ever changed the marshalling method of the Basic Formatted I/O interface, **THEN** that vendor's uninstaller **SHALL** re-register the VISA COM Type Library that is built into GlobMgr.dll.

RECOMMENDATION 6.2.3

It is strongly recommended that only the Universal marshaller be used for the Basic Formatted I/O interface.

RULE 6.2.7

Any vendor's COM component which marshals VISA COM-defined interfaces **SHALL** be installed with the Component Category ID "VISA COM I/O Custom Marshaller" with the GUID {db8cbf25-d6d3-11d4-aa51-00a024ee30bd}.

OBSERVATION 6.2.4

There are many possible side-effects of registering a marshaller other than the Universal Marshaller for the VISA COM interfaces, so to make problems caused by those installations more diagnosable and supportable, they need to be identifiable. This is a service that Component Category ID's provide.

PERMISSION 6.2.2

The Basic Formatted I/O Component **MAY** use the free-threaded marshaller for in-process marshalling.

6.2.3. Conflict Table Manager Component

The implementation of the Resource Conflict Manager component's main interface, IVisaConflictTableManager, is described in Section 4.4. There are a few additional requirements pertaining to the component itself.

RULE 6.2.8

The Resource Conflict Manager Component **SHALL** be thread-safe and runnable in single-threaded and multi-threaded apartments.

RULE 6.2.9

IF the Resource Conflict Manager version being installed is newer than the installed version on the target system, **THEN** the installer **SHALL** register the VISA COM Type Library that is built into GlobMgr.dll.

RECOMMENDATION 6.2.4

It is strongly recommended that the VISA COM Type Library not be installed as a separate TLB file. This will prevent version inconsistencies since the TLB file type is not versioned in a way that installers handle.

RULE 6.2.10

IF a vendor's installer or installed software ever changed the marshalling method of the Resource Conflict Manager interface, **THEN** that vendor's uninstaller **SHALL** re-register the VISA COM Type Library.

RECOMMENDATION 6.2.5

It is strongly recommended that only the Universal marshaller be used for the Resource Conflict Manager interface.

RULE 6.2.11

Any vendor's COM component which marshals VISA COM-defined interfaces **SHALL** be installed with the Component Category ID "VISA COM I/O Custom Marshaller" with the GUID {db8cbf25-d6d3-11d4-aa51-00a024ee30bd}.

OBSERVATION 6.2.5

There are many possible side-effects of registering a marshaller other than the Universal Marshaller for the VISA COM interfaces, so to make problems caused by those installations more diagnosable and supportable, they need to be identifiable. This is a service that Component Category ID's provide.

PERMISSION 6.2.3

The Resource Conflict Manager Component **MAY** use the free-threaded marshaller for in-process marshalling.

6.2.4. Vendor-Specific Resource Manager

Specifications for the required behavior of the IResourceManager and IResourceManager3 interfaces for Vendor-Specific Resource Managers (SRMs) are discussed in Section 4.2, *The Vendor-Specific Resource Manager Component*. There are a few additional requirements pertaining to the Component itself.

RULE 6.2.12

An SRM **SHALL** be thread-safe and runnable in single-threaded and multi-threaded apartments.

RULE 6.2.13

IF a vendor's installer or installed software ever changed the marshalling method of any VISA COM I/O defined interfaces, **THEN** that vendor's uninstaller **SHALL** re-register the VISA COM Type Library.

RECOMMENDATION 6.2.6

It is strongly recommended that only the Universal marshaller be used for the VISA COM I/O defined interfaces.

RULE 6.2.14

Any vendor's COM component which marshals VISA COM-defined interfaces **SHALL** be installed with the Component Category ID "VISA COM I/O Custom Marshaller" with the GUID {db8cbf25-d6d3-11d4-aa51-00a024ee30bd}.

OBSERVATION 6.2.6

There are many possible side-effects of registering a marshaller other than the Universal Marshaller for the VISA COM interfaces, so to make problems caused by those installations more diagnosable and supportable, they need to be identifiable. This is a service that Component Category ID's provide.

PERMISSION 6.2.4

An SRM **MAY** use the free-threaded marshaller for in-process marshalling.

RECOMMENDATION 6.2.7

It is recommended that SRMs be as lightweight as possible because the GRM will load all the installed SRMs on the system upon the first call to the FindRsrc() or Open() or ParseRsrc() method.

6.2.5. VISA COM I/O Resource Component

The implementation of the interfaces of VISA COM I/O resource components is laid out in Section 3, *VISA Resource Template and IVisaSession*, and Section 5, *VISA COM I/O Resource Classes*. There are some additional specifications regarding the COM Components themselves and implementation of custom types.

RULE 6.2.15

VISA COM I/O Resource Components **SHALL** be thread-safe and runnable in single-threaded and multi-threaded apartments and **SHALL** be registered as "Both" in the registry.

RULE 6.2.16

All VISA COM I/O Resource Components **SHALL** implement the COM Interfaces IVisaSession and IEventManager.

PERMISSION 6.2.5

VISA COM I/O Resource Components **MAY** use the free-threaded marshaller for in-process marshalling.

RECOMMENDATION 6.2.8

While the two base interfaces are the minimum to create a compliant resource, useful resources should also implement all of the IXMessage interfaces if they wish to appear as a SCPI message-based resource or the IXRegister interfaces if they wish to appear as a register-based instrument.

OBSERVATION 6.2.7

As is specified in Section 5, *VISA COM I/O Resource Classes*, if the HardwareInterfaceType and SessionType properties of the IVisaSession interface correspond to a VISA COM I/O-specified type, the component must implement the additional interfaces specified for that type. For example, GPIB INSTR resources must implement IMessage, IAsyncMessage, and IGpib in addition to the two base interfaces.

RULE 6.2.17

VISA COM I/O resources that implement custom resource types (resources with interface types not defined by VISA COM I/O) **SHALL** use interface type numbers in the range 0x5000-0x6FFF.

RECOMMENDATION 6.2.9

VISA COM I/O vendors should coordinate with the VXI*plug&play* consortium so that their custom resource type numbers do not overlap with other vendors' custom resource type numbers.

RULE 6.2.18

IF a VISA COM I/O Resource Component implements co-classes, **THEN** it **SHALL** also implement the COM interface `IProvideClassInfo2`.

OBSERVATION 6.2.8

This specification reserves the following range of GUID values, from `{DB8CBF00-D6D3-11D4-AA51-00A024EE30BD}` to `{DB8CC1FF-D6D3-11D4-AA51-00A024EE30BD}`. Interfaces and classes in this version of the specification use the range from `{DB8CBF00-D6D3-11D4-AA51-00A024EE30BD}` to `{DB8CBF2A-D6D3-11D4-AA51-00A024EE30BD}`. The other GUID values are reserved for future versions of this specification.

Section 7: Formatted I/O

Currently there is only one interface with formatted I/O services defined, the Basic Formatted I/O component and the IFormattedIO488 interface. This interface is designed to provide ease of use for the 95% case for formatted I/O needs with instruments that are 488.2 compliant or compatible. It is difficult to provide a complete formatted I/O solution as part of this standard due to the different formatted I/O paradigms of the various COM client platforms and the requirement that this be a freely distributable component.

7.1. IFormattedIO488 Interface

```
   [
      object,
      oleautomation,
      helpstring("IEEE 488.2 Formatted I/O Interface"),
      uuid(db8cbf1a-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIFormattedIO488 + 49),
      pointer_default(unique)
   ]
   interface IFormattedIO488 : IUnknown
   {
      typedef [public, helpstring("ASCII Data Types"), v1_enum]
      enum IEEEASCIIType {
         ASCIIType_I2 = 2,
         ASCIIType_I4 = 3,
         ASCIIType_R4 = 4,
         ASCIIType_R8 = 5,
         ASCIIType_BSTR = 8,
         ASCIIType_Any = 12,
         ASCIIType_UI1 = 17
      } IEEEASCIIType;
      typedef [public, helpstring("Binary Data Types"), v1_enum]
      enum IEEEBinaryType {
         BinaryType_I2 = 2,
         BinaryType_I4 = 3,
         BinaryType_R4 = 4,
         BinaryType_R8 = 5,
         BinaryType_UI1 = 17
      } IEEEBinaryType;

      [propget,helpstring("Get/Set the I/O Stream to
use"),helpcontext(HlpCtxIFormattedIO488 + 1)]
      HRESULT IO([out, retval] IMessage **pVal);
      [propputref,helpstring("Get/Set the I/O Stream to
use"),helpcontext(HlpCtxIFormattedIO488 + 1)]
      HRESULT IO([in] IMessage *newVal);
      [propget,helpstring("Get/Set whether the instrument communicates in Big
Endian (IEEE 488.2) format"),helpcontext(HlpCtxIFormattedIO488 + 2)]
      HRESULT InstrumentBigEndian([out, retval] VARIANT_BOOL *pVal);
      [propput,helpstring("Get/Set whether the instrument communicates in Big
Endian (IEEE 488.2) format"),helpcontext(HlpCtxIFormattedIO488 + 2)]
      HRESULT InstrumentBigEndian([in] VARIANT_BOOL newVal);

      [helpstring("Write a string to the I/O Stream and optionally flush the
buffer"),helpcontext(HlpCtxIFormattedIO488 + 3)]
      HRESULT WriteString(
         [in] BSTR data,
         [in, defaultvalue(TRUE)] VARIANT_BOOL flushAndEND);
      [helpstring("Write a single number to the I/O Stream and optionally flush
the buffer"),helpcontext(HlpCtxIFormattedIO488 + 4)]
      HRESULT WriteNumber(
         [in] VARIANT data,
         [in, defaultvalue(ASCIIType_Any)] IEEEASCIIType type,
         [in, defaultvalue(TRUE)] VARIANT_BOOL flushAndEND);
      [helpstring("Write a list of values to the I/O Stream and optionally flush
the buffer"),helpcontext(HlpCtxIFormattedIO488 + 5)]
      HRESULT WriteList(
         [in] VARIANT *data,
         [in, defaultvalue(ASCIIType_Any)] IEEEASCIIType type,
         [in, defaultvalue(",")] BSTR listSeperator,
         [in, defaultvalue(TRUE)] VARIANT_BOOL flushAndEND);
      [helpstring("Write a command followed by an IEEE 488.2 definite-length
binary block terminated with the Stream's termination character to the I/O
Stream"),helpcontext(HlpCtxIFormattedIO488 + 6)]
      HRESULT WriteIEEEBlock(
         [in] BSTR command,
```

```
        [in] VARIANT data,
        [in, defaultvalue(TRUE)] VARIANT_BOOL flushAndEND);
     [helpstring("Read the entire contents of the buffer until the termination
character / END signal and return the data as a
string"),helpcontext(HlpCtxIFormattedIO488 + 7)]
     HRESULT ReadString(
         [out, retval] BSTR *pData);
     [helpstring("Read a single number from the I/O Stream and optionally flush
the buffer"),helpcontext(HlpCtxIFormattedIO488 + 8)]
     HRESULT ReadNumber(
        [in, defaultvalue(ASCIIType_Any)] IEEEASCIIType type,
        [in, defaultvalue(TRUE)] VARIANT_BOOL flushToEND,
        [out, retval] VARIANT *pData);
     [helpstring("Read a list of values in ASCII format from the I/O Stream,
convert them to the specified type, and optionally flush the
buffer"),helpcontext(HlpCtxIFormattedIO488 + 9)]
     HRESULT ReadList(
        [in, defaultvalue(ASCIIType_Any)] IEEEASCIIType type,
        [in, defaultvalue(",;")] BSTR listSeperator,
        [out, retval] VARIANT *pData);
     [helpstring("Read a definite-length IEEE block from the I/O Stream and
optionally flush the buffer"),helpcontext(HlpCtxIFormattedIO488 + 10)]
     HRESULT ReadIEEEBlock(
        [in] IEEEBinaryType type,
        [in, defaultvalue(FALSE)] VARIANT_BOOL seekToBlock,
        [in, defaultvalue(TRUE)] VARIANT_BOOL flushToEND,
        [out, retval] VARIANT *pData);
     [helpstring("Flush the Write Buffer and optionally send the END
signal"),helpcontext(HlpCtxIFormattedIO488 + 11)]
     HRESULT FlushWrite(
         [in, defaultvalue(TRUE)] VARIANT_BOOL sendEND);
     [helpstring("Flush the Read Buffer"),helpcontext(HlpCtxIFormattedIO488 +
12)]
     HRESULT FlushRead();
     [helpstring("Set the formatted I/O read or write buffer
size"),helpcontext(HlpCtxIFormattedIO488 + 13)]
     HRESULT SetBufferSize(
        [in] enum BufferMask mask,
        [in] long size);
   };
```

RULE 7.1.1

All the methods of IFormattedIO488 **SHALL** return the error E_VISA_INV_SETUP when called before the property IO is properly set.

RULE 7.1.2

The putref property IO **SHALL** cause the component to hold a reference to the VISA COM I/O resource object passed in if it is a valid COM Object that implements the interface IMessage.

RULE 7.1.3

If the putref IO is successfully called after a previous successful call to the putref IO property on a formatted I/O object, the object **SHALL** flush all the buffers on the old reference and release the previously set reference.

RULE 7.1.4

The get IO property **SHALL** call AddRef on and return a reference to the IMessage Interface of the VISA COM I/O resource object it is holding.

RULE 7.1.5

The WriteString method **SHALL** convert the BSTR passed in to an ASCII string. If there is a Unicode character that has an ambiguous or no conversion to ASCII, the method **SHALL** quit and return the error E_VISA_INV_FMT.

RECOMMENDATION 7.1.1

Upon failure due to an invalid Unicode character, the method should place an IErrorInfo structure on the thread-local storage describing the problem and the character that caused the error.

RULE 7.1.6
The WriteString method **SHALL** add the C string it creates to its internal formatted I/O write buffer.

RULE 7.1.7
If any Write method is successfully called with the FlushAndEND parameter to true, it **SHALL** commit the write buffer after completing all other operations.

RULE 7.1.8
Committing the write buffer **SHALL** consist of creating a SAFEARRAY of bytes the size of the buffer and calling the IMessage interface reference's Write method with the array.

RULE 7.1.9
The WriteNumber method **SHALL** accept the following data types and convert them to ASCII characters using the decimal numeric data rules as proscribed by IEEE 488.2 and add the ASCII string to the formatted I/O write buffer.

VARIANT Type	IEEE ASCII Coding Format
VT_UI1	NR1
VT_I2	NR1
VT_I4	NR1
VT_R4	NR2
VT_R8	NR2

RULE 7.1.10
IF the DataType parameter is ASCIIType_Any, **AND** the actual type is any variant type other than those allowed, **THEN** WriteNumber **SHALL** fail and return E_INVALIDARG. **IF** the DataType parameter is any type other than those allowed, **THEN** WriteNumber **SHALL** fail and return E_INVALIDARG.

RULE 7.1.11
The WriteList method **SHALL** create an ASCII string containing the elements of the array arguments separated by the separator string passed in and add the string to the write buffer.

RULE 7.1.12
Any string arguments that have an ambiguous or no valid conversion to ASCII strings **SHALL** cause the WriteList method to fail and return E_VISA_INV_FMT.

RULE 7.1.13
The WriteList method **SHALL** accept SAFEARRAYs of specific data types given the DataType parameter's values and fail with the error code of E_INVALIDARG if the Data parameter is any other type according to this table.

List Argument Value	Valid "List" Parameter VARIANT types
ASCIIType_I2	VT_ARRAY | VT_I2
ASCIIType_I4	VT_ARRAY | VT_I4
ASCIIType_R4	VT_ARRAY | VT_R4
ASCIIType_R8	VT_ARRAY | VT_R8
ASCIIType_BSTR	VT_ARRAY | VT_BSTR
ASCIIType_Any	VT_ARRAY | VT_VARIANT
ASCIIType_UI1	VT_ARRAY | VT_UI1

RULE 7.1.14

WriteList, upon receiving a `DataType` argument of ASCIIType_Any **SHALL** return an error code of E_INVALIDARG if any of the VARIANT elements of the array argument `Data` are of types other than VT_UI1, VT_I2, VT_I4, VT_R4, VT_R8, and VT_BSTR, or a combination of VT_BYREF and those types.

RULE 7.1.15

WriteList **SHALL** convert the numeric arguments to ASCII strings as described for the WriteNumber method and convert BSTR arguments to strings as described for the WriteString method.

RULE 7.1.16

WriteList **SHALL** place the separator string between each element in the ASCII string it creates, but not at the beginning or end of the string.

RULE 7.1.17

If the ListSeperator argument of WriteList is empty, it **SHALL** use the comma ASCII character as the separator.

RULE 7.1.18

The WriteIEEEBlock method **SHALL** fail with the error code E_INVALIDARG if the argument "Data" is not a SAFEARRAY of numeric types.

RULE 7.1.19

The WriteIEEEBlock method **SHALL** perform conversions and place data into the write buffer according to this table (where <length> is calculated from the SAFEARRAY's data and the "Data" argument is assumed to be the c array equivalent of the SAFEARRAY's contents.)

Data Argument Variant Type and InstrumentBigEndian Value	Equivalent viPrintf Statement
VT_ARRAY | VT_UI, TRUE	viPrintf("%b<length>"), data)
VT_ARRAY | VT_I2, TRUE	viPrintf("%b<length>h"), data)
VT_ARRAY | VT_I4, TRUE	viPrintf("%b<length>l"), data)
VT_ARRAY | VT_R4, TRUE	viPrintf("%b<length>f"), data)
VT_ARRAY | VT_R8, TRUE	viPrintf("%b<length>d"), data)
VT_ARRAY | VT_UI, FALSE	No VISA equivalent
VT_ARRAY | VT_I2, FALSE	No VISA equivalent
VT_ARRAY | VT_I4, FALSE	No VISA equivalent
VT_ARRAY | VT_R4, FALSE	No VISA equivalent
VT_ARRAY | VT_R8, FALSE	No VISA equivalent

RULE 7.1.20

Whenever insufficient data is in the read buffer to complete one of the Formatted I/O read methods, the Formatted I/O object **SHALL** continue to call the Read method of the IMessage interface reference it holds until all the data it needs is retrieved, a timeout or other error occurs, or an END or termination character is received.

RULE 7.1.21

If a timeout occurs but enough data was retrieved to complete the request, the formatted I/O object **SHALL NOT** return an error.

RULE 7.1.22

IF the underlying read stops due to a termination character in the middle of an IEEE Block that is being parsed by ReadIEEEBlock, **THEN** ReadIEEEBlock **SHALL** continue retrieving data from the VISA COM I/O Resource object.

PERMISSION 7.1.1

The ReadIEEEBlock method **MAY** disable the termination character if it is enabled while calling Reads inside the (well-defined) length of the IEEE binary block, but must turn it back on before reading bytes lying outside the block.

OBSERVATION 7.1.1

A timeout can occur but the operation can still be successful if the END signal is suppressed and the termination character is disabled, in which case the only way to complete reading data of indefinite size is to encounter a timeout.

RULE 7.1.23

The ReadString method **SHALL** read from and remove characters from the formatted I/O read buffer until an END condition or termination character or an error occurs and convert the ASCII string to a BSTR and return it. ReadString **SHALL** return any error that occurs.

RULE 7.1.24

The ReadNumber method **SHALL** read from the read buffer as proscribed in section 7.7.2.2 of IEEE 488.2-1992 (but allow leading whitespace and stop upon the END signal or termination character) and convert the retrieved number to a VARIANT containing a VT_R8 and return the VARIANT. It **SHALL** remove the characters making up the number from the formatted I/O buffer.

RULE 7.1.25

The ReadNumber **SHALL** return all the characters retrieved to the buffer and return the error E_VISA_NSUP_FMT upon receiving a character that is not parsable.

RULE 7.1.26

The ReadList method **SHALL** return a VARIANT with the following types given the ASCIIType argument.

ASCIIType Value	Variant return value Type
ASCIIType_UI1	VT_ARRAY | VT_UI1
ASCIIType_I2	VT_ARRAY | VT_I2
ASCIIType_I4	VT_ARRAY | VT_I4
ASCIIType_R4	VT_ARRAY | VT_R4
ASCIIType_R8	VT_ARRAY | VT_R8
ASCIIType_BSTR	VT_ARRAY | VT_BSTR
ASCIIType_Any	VT_ARRAY | VT_VARIANT

RULE 7.1.27

Between the reading of each element, ReadList **SHALL** read and remove from the buffer whitespace and the separator string argument. If no separator is found before the first non-whitespace character or the END or termination character conditions, the method **SHALL** place that character into the buffer and return successfully.

RULE 7.1.28

ReadList **SHALL** read numeric types as described in the ReadNumber method.

RULE 7.1.29

ReadList **SHALL** return an error if the first non-whitespace character of BSTR arguments is not the quotation mark. ReadList **SHALL** complete reading the element and remove the characters from the buffer when a second quotation mark is read.

RULE 7.1.30

IF the ASCIIType value is ASCIITYPE_Any, **AND** the first non-whitespace characer is a quotation mark, **THEN** ReadList **SHALL** treat an element of a list as a string and create a VARIANT of type VT_BSTR for the returned SAFEARRAY.

RULE 7.1.31

IF the ASCIIType value is ASCIIType_Any, **AND** an NR1 parser that allows for leading whitespace successfully reads a number, **THEN** `ReadList()` **SHALL** treat an element of a list as a string and create a VARIANT of type VT_I4. Otherwise, **IF** an NR2 or NR3 parser successfully reads a number, **THEN** a VARIANT of type VT_R8 **SHALL** be created to hold it.

RULE 7.1.32

ReadList **SHALL** return an error of E_VISA_NSUP_FMT if no valid element type could be parsed when the ASCIIType argument is ASCIIType_Any.

RULE 7.1.33

IF the `SeekToBlock` parameter is true, **THEN** the ReadIEEEBlock method **SHALL** read and discard data until the hash '#' is encountered. The ReadIEEEBlock **SHALL** return the error E_VISA_NSUP_FMT if no hash is encountered or if SeekToBlock is false and the first character in the buffer is not a hash.

RULE 7.1.34

The ReadIEEEBlock method **SHALL** read data from the read buffer until it has completed reading the IEEE block according to this table.

Type argument value, Instrument Big Endian	Equivalent viScanf Statement
`BinaryType_UI1, true`	`viScanf(io, "%b", data)`
`BinaryType_I2, true`	`viScanf(io, "%bh", data)`
`BinaryType_I4, true`	`viScanf(io, "%bl", data)`
`BinaryType_R4, true`	`viScanf(io, "%bz", data)`
`BinaryType_R8, true`	`viScanf(io, "%bd", data)`
`BinaryType_UI1, false`	`No VISA equivalent`
`BinaryType_I2, false`	`No VISA equivalent`
`BinaryType_I4, false`	`No VISA equivalent`
`BinaryType_R4, false`	`No VISA equivalent`
`BinaryType_R8, false`	`No VISA equivalent`

OBSERVATION 7.1.2

The ReadIEEEBlock method will return successfully if if an END occurs before the expected block size is fully read. Several instruments set the size to be a maximum size, not necessarily the actual size, and then return END when they are done. In these situations, ReadIEEEBlock must not fail. This is directly opposite the behavior specified in a previous version of this specification, and the rule that had required an error condition on this behavior has been removed.

RULE 7.1.35

The FlushWrite method **SHALL** flush all the data to the cached VISA COM I/O Resource object.

RULE 7.1.36

IF the SendEND argument of FlushWrite is true, **THEN** the FlushWrite method **SHALL** send an END after flushing.

RULE 7.1.37

The FlushRead method **SHALL** discard any data in the read buffer until the END or termination character conditions.

RULE 7.1.38

Any implementation of `IFormattedIO488` **SHALL** also implement the COM interface `IProvideClassInfo2`.

RULE 7.1.39

The `ListSeparator` parameter to `ReadList()` **SHALL** be treated as a multi-character string of the type BSTR. Each character in the string **SHALL** be treated equally.

OBSERVATION 7.1.3

One way to treat characters in `ListSeparator` equally is to compare each character in the input data stream against the `ListSeparator` string using a function such as the ANSI C `strtok()`.

RULE 7.1.40

IF the `SendEND` parameter to `FlushWrite()` is FALSE, **THEN** the implementation **SHALL** disable the `SendEndEnabled` property on the I/O stream, commit the write buffer, and then restore the `SendEndEnabled` property.

OBSERVATION 7.1.4

If the user invokes a read method without previously having invoked a write method, the 488.2 rules specify that such an indiscriminate read is invalid. Rather than tracking this state in the `IFormattedIO488` implementation, the most likely scenario is that the lower-level IO will timeout. This will still generate an error, which should be expected in this case.

Section 8: The Complete VISA COM I/O IDL

There are two IDL files that comprise the types and interfaces in this specification. They are defined in the following two sections.

8.1. VisaCom.idl

Below is the complete IDL specification for VISA COM I/O.

```
//=============================================================================
//
//  Title     : VisaCom.idl
//  Platforms : Win32
//  Copyright : VXIplug&play Systems Alliance 2013.  All Rights Reserved.
//  Date      : 09-03-13
//
//=============================================================================

//=============================================================================
//  Type Library for VISA COM
//=============================================================================

[
  uuid(db8cbf00-d6d3-11d4-aa51-00a024ee30bd),
  version(5.2),
  helpstring("VISA COM 5.2 Type Library")
]
library VisaComLib
{
  importlib("stdole32.tlb");
  importlib("stdole2.tlb");

  #include <winerror.h>
  #include "visatype.idl"

  #define VISA_HRESULT(stat)                                                  \
    MAKE_HRESULT(                                                            \
       ((stat)&_VI_ERROR) ? SEVERITY_ERROR : SEVERITY_SUCCESS,              \
       FACILITY_ITF,                                                        \
       (stat)&0x0FFF)

//=============================================================================
// Enums
// These will show up as constants as well as enums in VB's object browser
// to provide easy access to the constants these enums contain.
//=============================================================================

  typedef [public, helpcontext(HlpCtxEnumVisaStatusCode), helpstring("VISA COM
Status Codes"), v1_enum]
  enum VisaStatusCode {
     [helpcontext(HlpCtxEnumVisaStatusCode + 1)] S_VISA_SUCCESS           = S_OK
,
     [helpcontext(HlpCtxEnumVisaStatusCode + 2)] S_VISA_EVENT_EN          =
VISA_HRESULT(VI_SUCCESS_EVENT_EN)         ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 3)] S_VISA_EVENT_DIS         =
VISA_HRESULT(VI_SUCCESS_EVENT_DIS)        ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 4)] S_VISA_QUEUE_EMPTY       =
VISA_HRESULT(VI_SUCCESS_QUEUE_EMPTY)      ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 5)] S_VISA_TERM_CHAR         =
VISA_HRESULT(VI_SUCCESS_TERM_CHAR)        ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 6)] S_VISA_MAX_CNT           =
VISA_HRESULT(VI_SUCCESS_MAX_CNT)          ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 7)] S_VISA_DEV_NPRESENT      =
VISA_HRESULT(VI_SUCCESS_DEV_NPRESENT)     ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 8)] S_VISA_QUEUE_NEMPTY      =
VISA_HRESULT(VI_SUCCESS_QUEUE_NEMPTY)     ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 9)] S_VISA_TRIG_MAPPED       =
VISA_HRESULT(VI_SUCCESS_TRIG_MAPPED)      ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 10)] S_VISA_NCHAIN            =
VISA_HRESULT(VI_SUCCESS_NCHAIN)           ,
```

```
      [helpcontext(HlpCtxEnumVisaStatusCode + 11)] S_VISA_NESTED_SHARED     =
VISA_HRESULT(VI_SUCCESS_NESTED_SHARED)    ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 12)] S_VISA_NESTED_EXCLUSIVE  =
VISA_HRESULT(VI_SUCCESS_NESTED_EXCLUSIVE) ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 13)] S_VISA_SYNC              =
VISA_HRESULT(VI_SUCCESS_SYNC)             ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 14)] S_VISA_QUEUE_OVERFLOW    =
VISA_HRESULT(VI_WARN_QUEUE_OVERFLOW)      ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 15)] S_VISA_CONFIG_NLOADED    =
VISA_HRESULT(VI_WARN_CONFIG_NLOADED)      ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 16)] S_VISA_NULL_OBJECT       =
VISA_HRESULT(VI_WARN_NULL_OBJECT)         ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 17)] S_VISA_NSUP_ATTR_STATE   =
VISA_HRESULT(VI_WARN_NSUP_ATTR_STATE)     ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 18)] S_VISA_UNKNOWN_STATUS    =
VISA_HRESULT(VI_WARN_UNKNOWN_STATUS)      ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 19)] S_VISA_NSUP_BUF          =
VISA_HRESULT(VI_WARN_NSUP_BUF)            ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 20)] S_VISA_EXT_FUNC_NIMPL    =
VISA_HRESULT(VI_WARN_EXT_FUNC_NIMPL)      ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 21)] E_VISA_SYSTEM_ERROR      =
VISA_HRESULT(VI_ERROR_SYSTEM_ERROR)       ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 22)] E_VISA_INV_OBJECT        =
VISA_HRESULT(VI_ERROR_INV_OBJECT)         ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 23)] E_VISA_RSRC_LOCKED       =
VISA_HRESULT(VI_ERROR_RSRC_LOCKED)        ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 24)] E_VISA_INV_EXPR          =
VISA_HRESULT(VI_ERROR_INV_EXPR)           ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 25)] E_VISA_RSRC_NFOUND       =
VISA_HRESULT(VI_ERROR_RSRC_NFOUND)        ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 26)] E_VISA_INV_RSRC_NAME     =
VISA_HRESULT(VI_ERROR_INV_RSRC_NAME)      ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 27)] E_VISA_INV_ACC_MODE      =
VISA_HRESULT(VI_ERROR_INV_ACC_MODE)       ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 28)] E_VISA_TMO               =
VISA_HRESULT(VI_ERROR_TMO)                ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 29)] E_VISA_CLOSING_FAILED    =
VISA_HRESULT(VI_ERROR_CLOSING_FAILED)     ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 30)] E_VISA_INV_DEGREE        =
VISA_HRESULT(VI_ERROR_INV_DEGREE)         ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 31)] E_VISA_INV_JOB_ID        =
VISA_HRESULT(VI_ERROR_INV_JOB_ID)         ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 32)] E_VISA_NSUP_ATTR         =
VISA_HRESULT(VI_ERROR_NSUP_ATTR)          ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 33)] E_VISA_NSUP_ATTR_STATE   =
VISA_HRESULT(VI_ERROR_NSUP_ATTR_STATE)    ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 34)] E_VISA_ATTR_READONLY     =
VISA_HRESULT(VI_ERROR_ATTR_READONLY)      ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 35)] E_VISA_INV_LOCK_TYPE     =
VISA_HRESULT(VI_ERROR_INV_LOCK_TYPE)      ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 36)] E_VISA_INV_ACCESS_KEY    =
VISA_HRESULT(VI_ERROR_INV_ACCESS_KEY)     ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 37)] E_VISA_INV_EVENT         =
VISA_HRESULT(VI_ERROR_INV_EVENT)          ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 38)] E_VISA_INV_MECH          =
VISA_HRESULT(VI_ERROR_INV_MECH)           ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 39)] E_VISA_HNDLR_NINSTALLED  =
VISA_HRESULT(VI_ERROR_HNDLR_NINSTALLED)   ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 40)] E_VISA_INV_HNDLR_REF     =
VISA_HRESULT(VI_ERROR_INV_HNDLR_REF)      ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 41)] E_VISA_INV_CONTEXT       =
VISA_HRESULT(VI_ERROR_INV_CONTEXT)        ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 42)] E_VISA_QUEUE_OVERFLOW    =
VISA_HRESULT(VI_ERROR_QUEUE_OVERFLOW)     ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 43)] E_VISA_NENABLED          =
VISA_HRESULT(VI_ERROR_NENABLED)           ,
      [helpcontext(HlpCtxEnumVisaStatusCode + 44)] E_VISA_ABORT             =
VISA_HRESULT(VI_ERROR_ABORT)              ,
```

```
    [helpcontext(HlpCtxEnumVisaStatusCode + 45)] E_VISA_RAW_WR_PROT_VIOL  =
VISA_HRESULT(VI_ERROR_RAW_WR_PROT_VIOL)   ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 46)] E_VISA_RAW_RD_PROT_VIOL  =
VISA_HRESULT(VI_ERROR_RAW_RD_PROT_VIOL)   ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 47)] E_VISA_OUTP_PROT_VIOL    =
VISA_HRESULT(VI_ERROR_OUTP_PROT_VIOL)     ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 48)] E_VISA_INP_PROT_VIOL     =
VISA_HRESULT(VI_ERROR_INP_PROT_VIOL)      ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 49)] E_VISA_BERR              =
VISA_HRESULT(VI_ERROR_BERR)               ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 50)] E_VISA_IN_PROGRESS       =
VISA_HRESULT(VI_ERROR_IN_PROGRESS)        ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 51)] E_VISA_INV_SETUP         =
VISA_HRESULT(VI_ERROR_INV_SETUP)          ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 52)] E_VISA_QUEUE_ERROR       =
VISA_HRESULT(VI_ERROR_QUEUE_ERROR)        ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 53)] E_VISA_ALLOC             =
VISA_HRESULT(VI_ERROR_ALLOC)              ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 54)] E_VISA_INV_MASK          =
VISA_HRESULT(VI_ERROR_INV_MASK)           ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 55)] E_VISA_IO                =
VISA_HRESULT(VI_ERROR_IO)                 ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 56)] E_VISA_INV_FMT           =
VISA_HRESULT(VI_ERROR_INV_FMT)            ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 57)] E_VISA_NSUP_FMT          =
VISA_HRESULT(VI_ERROR_NSUP_FMT)           ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 58)] E_VISA_LINE_IN_USE       =
VISA_HRESULT(VI_ERROR_LINE_IN_USE)        ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 59)] E_VISA_NSUP_MODE         =
VISA_HRESULT(VI_ERROR_NSUP_MODE)          ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 60)] E_VISA_SRQ_NOCCURRED     =
VISA_HRESULT(VI_ERROR_SRQ_NOCCURRED)      ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 61)] E_VISA_INV_SPACE         =
VISA_HRESULT(VI_ERROR_INV_SPACE)          ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 62)] E_VISA_INV_OFFSET        =
VISA_HRESULT(VI_ERROR_INV_OFFSET)         ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 63)] E_VISA_INV_WIDTH         =
VISA_HRESULT(VI_ERROR_INV_WIDTH)          ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 64)] E_VISA_NSUP_OFFSET       =
VISA_HRESULT(VI_ERROR_NSUP_OFFSET)        ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 65)] E_VISA_NSUP_VAR_WIDTH    =
VISA_HRESULT(VI_ERROR_NSUP_VAR_WIDTH)     ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 66)] E_VISA_WINDOW_NMAPPED    =
VISA_HRESULT(VI_ERROR_WINDOW_NMAPPED)     ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 67)] E_VISA_RESP_PENDING      =
VISA_HRESULT(VI_ERROR_RESP_PENDING)       ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 68)] E_VISA_NLISTENERS        =
VISA_HRESULT(VI_ERROR_NLISTENERS)         ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 69)] E_VISA_NCIC              =
VISA_HRESULT(VI_ERROR_NCIC)               ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 70)] E_VISA_NSYS_CNTLR        =
VISA_HRESULT(VI_ERROR_NSYS_CNTLR)         ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 71)] E_VISA_NSUP_OPER         =
VISA_HRESULT(VI_ERROR_NSUP_OPER)          ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 72)] E_VISA_INTR_PENDING      =
VISA_HRESULT(VI_ERROR_INTR_PENDING)       ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 73)] E_VISA_ASRL_PARITY       =
VISA_HRESULT(VI_ERROR_ASRL_PARITY)        ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 74)] E_VISA_ASRL_FRAMING      =
VISA_HRESULT(VI_ERROR_ASRL_FRAMING)       ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 75)] E_VISA_ASRL_OVERRUN      =
VISA_HRESULT(VI_ERROR_ASRL_OVERRUN)       ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 76)] E_VISA_TRIG_NMAPPED      =
VISA_HRESULT(VI_ERROR_TRIG_NMAPPED)       ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 77)] E_VISA_NSUP_ALIGN_OFFSET =
VISA_HRESULT(VI_ERROR_NSUP_ALIGN_OFFSET)  ,
    [helpcontext(HlpCtxEnumVisaStatusCode + 78)] E_VISA_USER_BUF          =
VISA_HRESULT(VI_ERROR_USER_BUF)           ,
```

```
     [helpcontext(HlpCtxEnumVisaStatusCode + 79)] E_VISA_RSRC_BUSY         =
VISA_HRESULT(VI_ERROR_RSRC_BUSY)          ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 80)] E_VISA_NSUP_WIDTH        =
VISA_HRESULT(VI_ERROR_NSUP_WIDTH)         ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 81)] E_VISA_INV_PARAMETER     =
VISA_HRESULT(VI_ERROR_INV_PARAMETER)      ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 82)] E_VISA_INV_PROT          =
VISA_HRESULT(VI_ERROR_INV_PROT)           ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 83)] E_VISA_INV_SIZE          =
VISA_HRESULT(VI_ERROR_INV_SIZE)           ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 84)] E_VISA_WINDOW_MAPPED     =
VISA_HRESULT(VI_ERROR_WINDOW_MAPPED)      ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 85)] E_VISA_NIMPL_OPER        =
VISA_HRESULT(VI_ERROR_NIMPL_OPER)         ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 86)] E_VISA_INV_LENGTH        =
VISA_HRESULT(VI_ERROR_INV_LENGTH)         ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 87)] E_VISA_INV_MODE          =
VISA_HRESULT(VI_ERROR_INV_MODE)           ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 88)] E_VISA_SESN_NLOCKED      =
VISA_HRESULT(VI_ERROR_SESN_NLOCKED)       ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 89)] E_VISA_MEM_NSHARED       =
VISA_HRESULT(VI_ERROR_MEM_NSHARED)        ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 90)] E_VISA_LIBRARY_NFOUND    =
VISA_HRESULT(VI_ERROR_LIBRARY_NFOUND)     ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 91)] E_VISA_NSUP_INTR         =
VISA_HRESULT(VI_ERROR_NSUP_INTR)          ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 92)] E_VISA_INV_LINE          =
VISA_HRESULT(VI_ERROR_INV_LINE)           ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 93)] E_VISA_FILE_ACCESS       =
VISA_HRESULT(VI_ERROR_FILE_ACCESS)        ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 94)] E_VISA_FILE_IO           =
VISA_HRESULT(VI_ERROR_FILE_IO)            ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 95)] E_VISA_NSUP_LINE         =
VISA_HRESULT(VI_ERROR_NSUP_LINE)          ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 96)] E_VISA_NSUP_MECH         =
VISA_HRESULT(VI_ERROR_NSUP_MECH)          ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 97)] E_VISA_INTF_NUM_NCONFIG  =
VISA_HRESULT(VI_ERROR_INTF_NUM_NCONFIG)   ,
     [helpcontext(HlpCtxEnumVisaStatusCode + 98)] E_VISA_CONN_LOST         =
VISA_HRESULT(VI_ERROR_CONN_LOST)
  } VisaStatusCode;

  typedef [public, helpcontext(HlpCtxEnumEventType), helpstring("Event Type
Constants"), v1_enum]
  enum EventType {
     [helpcontext(HlpCtxEnumEventType + 1)] EVENT_IO_COMPLETION      =
VI_EVENT_IO_COMPLETION      ,
     [helpcontext(HlpCtxEnumEventType + 2)] EVENT_TRIG               =
VI_EVENT_TRIG               ,
     [helpcontext(HlpCtxEnumEventType + 3)] EVENT_SERVICE_REQ        =
VI_EVENT_SERVICE_REQ        ,
     [helpcontext(HlpCtxEnumEventType + 4)] EVENT_CLEAR              =
VI_EVENT_CLEAR              ,
     [helpcontext(HlpCtxEnumEventType + 5)] EVENT_EXCEPTION          =
VI_EVENT_EXCEPTION          ,
     [helpcontext(HlpCtxEnumEventType + 6)] EVENT_GPIB_CIC           =
VI_EVENT_GPIB_CIC           ,
     [helpcontext(HlpCtxEnumEventType + 7)] EVENT_GPIB_TALK          =
VI_EVENT_GPIB_TALK          ,
     [helpcontext(HlpCtxEnumEventType + 8)] EVENT_GPIB_LISTEN        =
VI_EVENT_GPIB_LISTEN        ,
     [helpcontext(HlpCtxEnumEventType + 9)] EVENT_VXI_VME_SYSFAIL    =
VI_EVENT_VXI_VME_SYSFAIL    ,
     [helpcontext(HlpCtxEnumEventType + 10)] EVENT_VXI_VME_SYSRESET   =
VI_EVENT_VXI_VME_SYSRESET   ,
     [helpcontext(HlpCtxEnumEventType + 11)] EVENT_VXI_SIGP           =
VI_EVENT_VXI_SIGP           ,
     [helpcontext(HlpCtxEnumEventType + 12)] EVENT_VXI_VME_INTR       =
```

```
VI_EVENT_VXI_VME_INTR       ,
     [helpcontext(HlpCtxEnumEventType + 13)] EVENT_TCPIP_CONNECT      =
VI_EVENT_TCPIP_CONNECT      ,
     [helpcontext(HlpCtxEnumEventType + 14)] EVENT_USB_INTR           =
VI_EVENT_USB_INTR           ,
     [helpcontext(HlpCtxEnumEventType + 15)] ALL_ENABLED_EVENTS       =
VI_ALL_ENABLED_EVENTS       ,
     [helpcontext(HlpCtxEnumEventType + 16)] CUSTOM_EVENT_TYPE        = -1
   } EventType;

   typedef [public, helpcontext(HlpCtxEnumHardwareInterfaceType),
helpstring("Hardware Interface Type Constants")]
   enum HardwareInterfaceType {
      [helpcontext(HlpCtxEnumHardwareInterfaceType + 1)] INTF_GPIB =
VI_INTF_GPIB,
      [helpcontext(HlpCtxEnumHardwareInterfaceType + 2)] INTF_VXI = VI_INTF_VXI,
      [helpcontext(HlpCtxEnumHardwareInterfaceType + 3)] INTF_GPIB_VXI =
VI_INTF_GPIB_VXI,
      [helpcontext(HlpCtxEnumHardwareInterfaceType + 4)] INTF_ASRL =
VI_INTF_ASRL,
      [helpcontext(HlpCtxEnumHardwareInterfaceType + 5)] INTF_TCPIP =
VI_INTF_TCPIP,
      [helpcontext(HlpCtxEnumHardwareInterfaceType + 6)] INTF_USB = VI_INTF_USB
   } HardwareInterfaceType;

   typedef [public, helpcontext(HlpCtxEnumIOProtocol), helpstring("I/O Protocol
Type Constants")]
   enum IOProtocol {
      [helpcontext(HlpCtxEnumIOProtocol + 1)] PROT_NORMAL = VI_PROT_NORMAL,
      [helpcontext(HlpCtxEnumIOProtocol + 2)] PROT_FDC = VI_PROT_FDC,
      [helpcontext(HlpCtxEnumIOProtocol + 3)] PROT_HS488 = VI_PROT_HS488,
      [helpcontext(HlpCtxEnumIOProtocol + 4)] PROT_4882_STRS = VI_PROT_4882_STRS,
      [helpcontext(HlpCtxEnumIOProtocol + 5)] PROT_USBTMC_VENDOR =
VI_PROT_USBTMC_VENDOR
   } IOProtocol;

   typedef [public, helpcontext(HlpCtxEnumFDCMode), helpstring("FDC Mode
Constants")]
   enum FDCMode {
      [helpcontext(HlpCtxEnumFDCMode + 1)] FDC_NORMAL = VI_FDC_NORMAL,
      [helpcontext(HlpCtxEnumFDCMode + 2)] FDC_STREAM = VI_FDC_STREAM
   } FDCMode;

   typedef [public, helpcontext(HlpCtxEnumAddressSpace), helpstring("Address
Space Constants")]
   enum AddressSpace {
      [helpcontext(HlpCtxEnumAddressSpace + 1)] LOCAL_SPACE = VI_LOCAL_SPACE,
      [helpcontext(HlpCtxEnumAddressSpace + 2)] VXI_A16_SPACE = VI_A16_SPACE,
      [helpcontext(HlpCtxEnumAddressSpace + 3)] VXI_A24_SPACE = VI_A24_SPACE,
      [helpcontext(HlpCtxEnumAddressSpace + 4)] VXI_A32_SPACE = VI_A32_SPACE,
      [helpcontext(HlpCtxEnumAddressSpace + 5)] OPAQUE_SPACE = VI_OPAQUE_SPACE
   } AddressSpace;

   typedef [public, helpcontext(HlpCtxEnumEventMechanism), helpstring("Event
Mechanism Constants")]
   enum EventMechanism {
      [helpcontext(HlpCtxEnumEventMechanism + 1)] EVENT_QUEUE = VI_QUEUE,
      [helpcontext(HlpCtxEnumEventMechanism + 2)] EVENT_HNDLR = VI_HNDLR,
      [helpcontext(HlpCtxEnumEventMechanism + 3)] EVENT_SUSPEND_HNDLR =
VI_SUSPEND_HNDLR,
      [helpcontext(HlpCtxEnumEventMechanism + 4)] EVENT_ALL_MECH = VI_ALL_MECH
   } EventMechanism;

   typedef [public, helpcontext(HlpCtxEnumTriggerLine), helpstring("Trigger Line
Constants")]
   enum TriggerLine {
      [helpcontext(HlpCtxEnumTriggerLine + 1)] TRIG_ALL = VI_TRIG_ALL,
      [helpcontext(HlpCtxEnumTriggerLine + 2)] TRIG_SW = VI_TRIG_SW,
```

```
      [helpcontext(HlpCtxEnumTriggerLine + 3)] TRIG_TTL0 = VI_TRIG_TTL0,
      [helpcontext(HlpCtxEnumTriggerLine + 4)] TRIG_TTL1 = VI_TRIG_TTL1,
      [helpcontext(HlpCtxEnumTriggerLine + 5)] TRIG_TTL2 = VI_TRIG_TTL2,
      [helpcontext(HlpCtxEnumTriggerLine + 6)] TRIG_TTL3 = VI_TRIG_TTL3,
      [helpcontext(HlpCtxEnumTriggerLine + 7)] TRIG_TTL4 = VI_TRIG_TTL4,
      [helpcontext(HlpCtxEnumTriggerLine + 8)] TRIG_TTL5 = VI_TRIG_TTL5,
      [helpcontext(HlpCtxEnumTriggerLine + 9)] TRIG_TTL6 = VI_TRIG_TTL6,
      [helpcontext(HlpCtxEnumTriggerLine + 10)] TRIG_TTL7 = VI_TRIG_TTL7,
      [helpcontext(HlpCtxEnumTriggerLine + 11)] TRIG_ECL0 = VI_TRIG_ECL0,
      [helpcontext(HlpCtxEnumTriggerLine + 12)] TRIG_ECL1 = VI_TRIG_ECL1,
      [helpcontext(HlpCtxEnumTriggerLine + 13)] TRIG_PANEL_IN = VI_TRIG_PANEL_IN,
      [helpcontext(HlpCtxEnumTriggerLine + 14)] TRIG_PANEL_OUT =
VI_TRIG_PANEL_OUT
   } TriggerLine;

   typedef [public, helpcontext(HlpCtxEnumTriggerProtocol), helpstring("Trigger
Protocol Constants")]
   enum TriggerProtocol {
      [helpcontext(HlpCtxEnumTriggerProtocol + 1)] TRIG_PROT_DEFAULT =
VI_TRIG_PROT_DEFAULT,
      [helpcontext(HlpCtxEnumTriggerProtocol + 2)] TRIG_PROT_ON =
VI_TRIG_PROT_ON,
      [helpcontext(HlpCtxEnumTriggerProtocol + 3)] TRIG_PROT_OFF =
VI_TRIG_PROT_OFF,
      [helpcontext(HlpCtxEnumTriggerProtocol + 4)] TRIG_PROT_SYNC =
VI_TRIG_PROT_SYNC
   } TriggerProtocol;

   typedef [public, helpcontext(HlpCtxEnumBufferMask), helpstring("Buffer Mask
Constants")]
   enum BufferMask {
      [helpcontext(HlpCtxEnumBufferMask + 1)] IO_IN_BUF = VI_IO_IN_BUF,
      [helpcontext(HlpCtxEnumBufferMask + 2)] IO_OUT_BUF = VI_IO_OUT_BUF,
      [helpcontext(HlpCtxEnumBufferMask + 3)] IO_IN_AND_OUT_BUFS =
VI_IO_IN_BUF|VI_IO_OUT_BUF
   } BufferMask;

   typedef [public, helpcontext(HlpCtxEnumTimeout), helpstring("Timeout
Constants"), v1_enum ]
   enum Timeout {
      [helpcontext(HlpCtxEnumTimeout + 1)] TMO_IMMEDIATE = VI_TMO_IMMEDIATE,
      [helpcontext(HlpCtxEnumTimeout + 2)] TMO_INFINITE = VI_TMO_INFINITE
   } Timeout;

   typedef [public, helpcontext(HlpCtxEnumAccessMode), helpstring("Access Mode
Constants")]
   enum AccessMode {
      [helpcontext(HlpCtxEnumAccessMode + 1)] NO_LOCK = VI_NO_LOCK,
      [helpcontext(HlpCtxEnumAccessMode + 2)] EXCLUSIVE_LOCK = VI_EXCLUSIVE_LOCK,
      [helpcontext(HlpCtxEnumAccessMode + 3)] SHARED_LOCK = VI_SHARED_LOCK,
      [helpcontext(HlpCtxEnumAccessMode + 4)] LOAD_CONFIG = VI_LOAD_CONFIG
   } AccessMode;

   typedef [public, helpcontext(HlpCtxEnumSerialParity), helpstring("Serial
Parity Constants")]
   enum SerialParity {
      [helpcontext(HlpCtxEnumSerialParity + 1)] ASRL_PAR_NONE = VI_ASRL_PAR_NONE,
      [helpcontext(HlpCtxEnumSerialParity + 2)] ASRL_PAR_ODD = VI_ASRL_PAR_ODD,
      [helpcontext(HlpCtxEnumSerialParity + 3)] ASRL_PAR_EVEN = VI_ASRL_PAR_EVEN,
      [helpcontext(HlpCtxEnumSerialParity + 4)] ASRL_PAR_MARK = VI_ASRL_PAR_MARK,
      [helpcontext(HlpCtxEnumSerialParity + 5)] ASRL_PAR_SPACE =
VI_ASRL_PAR_SPACE
   } SerialParity;

   typedef [public, helpcontext(HlpCtxEnumSerialStopBits), helpstring("Serial
Stop Bit Constants")]
   enum SerialStopBits {
      [helpcontext(HlpCtxEnumSerialStopBits + 1)] ASRL_STOP_ONE =
```

```
VI_ASRL_STOP_ONE,
     [helpcontext(HlpCtxEnumSerialStopBits + 2)] ASRL_STOP_ONE5 =
VI_ASRL_STOP_ONE5,
     [helpcontext(HlpCtxEnumSerialStopBits + 3)] ASRL_STOP_TWO =
VI_ASRL_STOP_TWO
   } SerialStopBits;

   typedef [public, helpcontext(HlpCtxEnumSerialFlowControl), helpstring("Serial
Flow Control Constants")]
   enum SerialFlowControl {
      [helpcontext(HlpCtxEnumSerialFlowControl + 1)] ASRL_FLOW_NONE =
VI_ASRL_FLOW_NONE,
      [helpcontext(HlpCtxEnumSerialFlowControl + 2)] ASRL_FLOW_XON_XOFF =
VI_ASRL_FLOW_XON_XOFF,
      [helpcontext(HlpCtxEnumSerialFlowControl + 3)] ASRL_FLOW_RTS_CTS =
VI_ASRL_FLOW_RTS_CTS,
      [helpcontext(HlpCtxEnumSerialFlowControl + 4)] ASRL_FLOW_DTR_DSR =
VI_ASRL_FLOW_DTR_DSR,
      [helpcontext(HlpCtxEnumSerialFlowControl + 5)]
ASRL_FLOW_RTS_CTS_AND_XON_XOFF = VI_ASRL_FLOW_RTS_CTS|VI_ASRL_FLOW_XON_XOFF,
      [helpcontext(HlpCtxEnumSerialFlowControl + 6)]
ASRL_FLOW_DTR_DSR_AND_XON_XOFF = VI_ASRL_FLOW_DTR_DSR|VI_ASRL_FLOW_XON_XOFF
   } SerialFlowControl;

   typedef [public, helpcontext(HlpCtxEnumSerialEndConst), helpstring("Serial END
Indicator Constants")]
   enum SerialEndConst {
      [helpcontext(HlpCtxEnumSerialEndConst + 1)] ASRL_END_NONE =
VI_ASRL_END_NONE,
      [helpcontext(HlpCtxEnumSerialEndConst + 2)] ASRL_END_LAST_BIT =
VI_ASRL_END_LAST_BIT,
      [helpcontext(HlpCtxEnumSerialEndConst + 3)] ASRL_END_TERMCHAR =
VI_ASRL_END_TERMCHAR,
      [helpcontext(HlpCtxEnumSerialEndConst + 4)] ASRL_END_BREAK =
VI_ASRL_END_BREAK
   } SerialEndConst;

   typedef [public, helpcontext(HlpCtxEnumLineState), helpstring("Digital Line
State Constants")]
   enum LineState {
      [helpcontext(HlpCtxEnumLineState + 1)] STATE_ASSERTED = VI_STATE_ASSERTED,
      [helpcontext(HlpCtxEnumLineState + 2)] STATE_UNASSERTED =
VI_STATE_UNASSERTED,
      [helpcontext(HlpCtxEnumLineState + 3)] STATE_UNKNOWN = VI_STATE_UNKNOWN
   } LineState;

   typedef [public, helpcontext(HlpCtxEnumVXIMemoryAccessPrivilege),
helpstring("VXI Memory Access Privilege Constants")]
   enum VXIMemoryAccessPrivilege {
      [helpcontext(HlpCtxEnumVXIMemoryAccessPrivilege + 1)] DATA_PRIV =
VI_DATA_PRIV,
      [helpcontext(HlpCtxEnumVXIMemoryAccessPrivilege + 2)] DATA_NPRIV =
VI_DATA_NPRIV,
      [helpcontext(HlpCtxEnumVXIMemoryAccessPrivilege + 3)] PROG_PRIV =
VI_PROG_PRIV,
      [helpcontext(HlpCtxEnumVXIMemoryAccessPrivilege + 4)] PROG_NPRIV =
VI_PROG_NPRIV,
      [helpcontext(HlpCtxEnumVXIMemoryAccessPrivilege + 5)] BLCK_PRIV =
VI_BLCK_PRIV,
      [helpcontext(HlpCtxEnumVXIMemoryAccessPrivilege + 6)] BLCK_NPRIV =
VI_BLCK_NPRIV,
      [helpcontext(HlpCtxEnumVXIMemoryAccessPrivilege + 7)] D64_PRIV =
VI_D64_PRIV,
      [helpcontext(HlpCtxEnumVXIMemoryAccessPrivilege + 8)] D64_NPRIV =
VI_D64_NPRIV
   } VXIMemoryAccessPrivilege;

   typedef [public, helpcontext(HlpCtxEnumDataWidth), helpstring("Data Transfer
```

```
Width Constants")]
   enum DataWidth {
      [helpcontext(HlpCtxEnumDataWidth + 1)] WIDTH_8 = VI_WIDTH_8,
      [helpcontext(HlpCtxEnumDataWidth + 2)] WIDTH_16 = VI_WIDTH_16,
      [helpcontext(HlpCtxEnumDataWidth + 3)] WIDTH_32 = VI_WIDTH_32
   } DataWidth;

   typedef [public, helpcontext(HlpCtxEnumRENControlConst), helpstring("GPIB REN
Control Constants")]
   enum RENControlConst {
      [helpcontext(HlpCtxEnumRENControlConst + 1)] GPIB_REN_DEASSERT =
VI_GPIB_REN_DEASSERT,
      [helpcontext(HlpCtxEnumRENControlConst + 2)] GPIB_REN_ASSERT =
VI_GPIB_REN_ASSERT,
      [helpcontext(HlpCtxEnumRENControlConst + 3)] GPIB_REN_GTL_AND_DEASSERT =
VI_GPIB_REN_DEASSERT_GTL,
      [helpcontext(HlpCtxEnumRENControlConst + 4)] GPIB_REN_ASSERT_AND_ADDRESS =
VI_GPIB_REN_ASSERT_ADDRESS,
      [helpcontext(HlpCtxEnumRENControlConst + 5)] GPIB_REN_LLO =
VI_GPIB_REN_ASSERT_LLO,
      [helpcontext(HlpCtxEnumRENControlConst + 6)] GPIB_REN_ADDRESS_AND_LLO =
VI_GPIB_REN_ASSERT_ADDRESS_LLO,
      [helpcontext(HlpCtxEnumRENControlConst + 7)] GPIB_REN_GTL =
VI_GPIB_REN_ADDRESS_GTL
   } RENControlConst;

   typedef [public, helpcontext(HlpCtxEnumATNControlConst), helpstring("GPIB ATN
Control Constants")]
   enum ATNControlConst {
      [helpcontext(HlpCtxEnumATNControlConst + 1)] GPIB_ATN_DEASSERT =
VI_GPIB_ATN_DEASSERT,
      [helpcontext(HlpCtxEnumATNControlConst + 2)] GPIB_ATN_ASSERT =
VI_GPIB_ATN_ASSERT,
      [helpcontext(HlpCtxEnumATNControlConst + 3)] GPIB_ATN_DEASSERT_HANDSHAKE =
VI_GPIB_ATN_DEASSERT_HANDSHAKE,
      [helpcontext(HlpCtxEnumATNControlConst + 4)] GPIB_ATN_ASSERT_IMMEDIATE =
VI_GPIB_ATN_ASSERT_IMMEDIATE
   } ATNControlConst;

   typedef [public, helpcontext(HlpCtxEnumGPIBAddressState), helpstring("GPIB
Addressing State Constants")]
   enum GPIBAddressState {
      [helpcontext(HlpCtxEnumGPIBAddressState + 1)] GPIB_UNADDRESSED =
VI_GPIB_UNADDRESSED,
      [helpcontext(HlpCtxEnumGPIBAddressState + 2)] GPIB_TALKER = VI_GPIB_TALKER,
      [helpcontext(HlpCtxEnumGPIBAddressState + 3)] GPIB_LISTENER =
VI_GPIB_LISTENER
   } GPIBAddressState;

   typedef [public, helpcontext(HlpCtxEnumVXICommandQuery), helpstring("VXI
Command Query Constants")]
   enum VXICommandQuery {
      [helpcontext(HlpCtxEnumVXICommandQuery + 1)] VXI_CMD16 = VI_VXI_CMD16,
      [helpcontext(HlpCtxEnumVXICommandQuery + 2)] VXI_CMD16_RESP16 =
VI_VXI_CMD16_RESP16,
      [helpcontext(HlpCtxEnumVXICommandQuery + 3)] VXI_RESP16 = VI_VXI_RESP16,
      [helpcontext(HlpCtxEnumVXICommandQuery + 4)] VXI_CMD32 = VI_VXI_CMD32,
      [helpcontext(HlpCtxEnumVXICommandQuery + 5)] VXI_CMD32_RESP16 =
VI_VXI_CMD32_RESP16,
      [helpcontext(HlpCtxEnumVXICommandQuery + 6)] VXI_CMD32_RESP32 =
VI_VXI_CMD32_RESP32,
      [helpcontext(HlpCtxEnumVXICommandQuery + 7)] VXI_RESP32 = VI_VXI_RESP32
   } VXICommandQuery;

   typedef [public, helpcontext(HlpCtxEnumAssertInterruptConst),
helpstring("Assert Interrupt Signal Constants")]
   enum AssertInterruptConst {
      [helpcontext(HlpCtxEnumAssertInterruptConst + 1)] ASSERT_SIGNAL =
```

```
VI_ASSERT_SIGNAL,
     [helpcontext(HlpCtxEnumAssertInterruptConst + 2)] ASSERT_USE_ASSIGNED =
VI_ASSERT_USE_ASSIGNED,
     [helpcontext(HlpCtxEnumAssertInterruptConst + 3)] ASSERT_IRQ1 =
VI_ASSERT_IRQ1,
     [helpcontext(HlpCtxEnumAssertInterruptConst + 4)] ASSERT_IRQ2 =
VI_ASSERT_IRQ2,
     [helpcontext(HlpCtxEnumAssertInterruptConst + 5)] ASSERT_IRQ3 =
VI_ASSERT_IRQ3,
     [helpcontext(HlpCtxEnumAssertInterruptConst + 6)] ASSERT_IRQ4 =
VI_ASSERT_IRQ4,
     [helpcontext(HlpCtxEnumAssertInterruptConst + 7)] ASSERT_IRQ5 =
VI_ASSERT_IRQ5,
     [helpcontext(HlpCtxEnumAssertInterruptConst + 8)] ASSERT_IRQ6 =
VI_ASSERT_IRQ6,
     [helpcontext(HlpCtxEnumAssertInterruptConst + 9)] ASSERT_IRQ7 =
VI_ASSERT_IRQ7
   } AssertInterruptConst;

   typedef [public, helpcontext(HlpCtxEnumAssertUtilityConst), helpstring("Assert
Utility Signal Constants")]
   enum AssertUtilityConst {
      [helpcontext(HlpCtxEnumAssertUtilityConst + 1)] ASSERT_SYSRESET =
VI_UTIL_ASSERT_SYSRESET,
     [helpcontext(HlpCtxEnumAssertUtilityConst + 2)] ASSERT_SYSFAIL =
VI_UTIL_ASSERT_SYSFAIL,
     [helpcontext(HlpCtxEnumAssertUtilityConst + 3)] DEASSERT_SYSFAIL =
VI_UTIL_DEASSERT_SYSFAIL
   } AssertUtilityConst;

   typedef [public, helpcontext(HlpCtxEnumVXIDevClass), helpstring("VXI Device
Class Constants")]
   enum VXIDevClass {
      [helpcontext(HlpCtxEnumVXIDevClass + 1)] VXI_CLASS_MEMORY =
VI_VXI_CLASS_MEMORY,
     [helpcontext(HlpCtxEnumVXIDevClass + 2)] VXI_CLASS_EXTENDED =
VI_VXI_CLASS_EXTENDED,
     [helpcontext(HlpCtxEnumVXIDevClass + 3)] VXI_CLASS_MESSAGE =
VI_VXI_CLASS_MESSAGE,
     [helpcontext(HlpCtxEnumVXIDevClass + 4)] VXI_CLASS_REGISTER =
VI_VXI_CLASS_REGISTER,
     [helpcontext(HlpCtxEnumVXIDevClass + 5)] VXI_CLASS_OTHER =
VI_VXI_CLASS_OTHER
   } VXIDevClass;

   typedef [public, helpcontext(HlpCtxEnumPXIMemType), helpstring("PXI Memory
Type Constants")]
   enum PXIMemType {
      [helpcontext(HlpCtxEnumPXIMemType + 1)] PXI_ADDR_NONE = VI_PXI_ADDR_NONE,
      [helpcontext(HlpCtxEnumPXIMemType + 2)] PXI_ADDR_MEM = VI_PXI_ADDR_MEM,
      [helpcontext(HlpCtxEnumPXIMemType + 3)] PXI_ADDR_IO = VI_PXI_ADDR_IO,
   } PXIMemType;

//=============================================================================
// Interfaces
//=============================================================================

   interface IVisaSession;    // Forward reference
   interface IEventManager;   // Forward reference

//=============================================================================
//  VISA Sesssion Management
//=============================================================================

   //-------------------------------------------------------------------------
   //  IResourceManager (obsolete)
   //-------------------------------------------------------------------------
   [
```

```
      object,
      oleautomation,
      helpstring("VISA Resource Manager Interface (obsolete)"),
      uuid(db8cbf02-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIResourceManager + 49),
      pointer_default(unique),
      hidden
   ]
   interface IResourceManager : IUnknown
   {
      [propget,helpstring("Get the manufacturer name of the
component"),helpcontext(HlpCtxIResourceManager + 1)]
      HRESULT SoftwareManufacturerName([out, retval] BSTR *pVal);
      [propget,helpstring("Get the manufacturer ID of the
component"),helpcontext(HlpCtxIResourceManager + 2)]
      HRESULT SoftwareManufacturerID([out, retval] short *pVal);
      [propget,helpstring("Get the description of the
component"),helpcontext(HlpCtxIResourceManager + 3)]
      HRESULT Description([out, retval] BSTR *pDesc);
      [propget,helpstring("Get the implementation version of the
component"),helpcontext(HlpCtxIResourceManager + 4)]
      HRESULT ComponentVersion([out, retval] long *pVal);
      [propget,helpstring("Get the ProgID of the
component"),helpcontext(HlpCtxIResourceManager + 5)]
      HRESULT ProgID([out, retval] BSTR *pVal);
      [propget,helpstring("Get the VISA COM I/O specification
version"),helpcontext(HlpCtxIResourceManager + 6)]
      HRESULT SpecVersion([out, retval] long *pVal);

      [helpstring("Find a list of resources that match a search
string"),helpcontext(HlpCtxIResourceManager + 7)]
      HRESULT FindRsrc(
         [in] BSTR expr,
         [out, retval] SAFEARRAY(BSTR) *pFindList);
      [helpstring("Initialize a session to the specified resource
name"),helpcontext(HlpCtxIResourceManager + 9)]
      HRESULT Open(
         [in] BSTR resourceName,
         [in, defaultvalue(NO_LOCK)] AccessMode mode,
         [in, defaultvalue(2000)] long openTimeout,
         [in, defaultvalue("")] BSTR optionString,
         [out, retval] IVisaSession **vi);
      [helpstring("Determine the validity and interface information of a resource
name"),helpcontext(HlpCtxIResourceManager + 10)]
      HRESULT ParseRsrc(
         [in] BSTR resourceName,
         [in, out] short *pInterfaceType,
         [in, out] short *pInterfaceNumber,
         [in, out] BSTR *pSessionType);
   };

   //---------------------------------------------------------------------------
   //  IResourceManager3
   //---------------------------------------------------------------------------
   [
      object,
      oleautomation,
      helpstring("VISA Resource Manager Interface"),
      uuid(db8cbf20-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIResourceManager3 + 49),
      pointer_default(unique)
   ]
   interface IResourceManager3 : IResourceManager
   {
      [helpstring("Determine the validity and interface information of a resource
name"),helpcontext(HlpCtxIResourceManager3 + 1)]
      HRESULT ParseRsrcEx(
         [in] BSTR resourceName,
```

```
        [in, out] short *pInterfaceType,
        [in, out] short *pInterfaceNumber,
        [in, out] BSTR *pSessionType,
        [in, out] BSTR *pUnaliasedExpandedResourceName,
        [in, out] BSTR *pAliasIfExists);
   };

//==============================================================================
//  VISA I/O Sessions
//==============================================================================

   //--------------------------------------------------------------------------
   //  IVisaSession
   //--------------------------------------------------------------------------
   [
      object,
      oleautomation,
      helpstring("VISA Session Interface"),
      uuid(db8cbf03-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIVisaSession + 49),
      pointer_default(unique)
   ]
   interface IVisaSession : IUnknown
   {
      [propget, helpcontext(HlpCtxIVisaSession  + 1), helpstring("Get the
implementation version of the component")]
      HRESULT ComponentVersion([out, retval] long *pVal);
      [propget, helpcontext(HlpCtxIVisaSession  + 2), helpstring("Get the VISA
COM I/O specification version")]
      HRESULT SpecVersion([out, retval] long *pVal);
      [propget, helpcontext(HlpCtxIVisaSession  + 3), helpstring("Get a
description of the hardware interface")]
      HRESULT HardwareInterfaceName([out, retval] BSTR *pVal);
      [propget, helpcontext(HlpCtxIVisaSession  + 4), helpstring("Get the
hardware interface number")]
      HRESULT HardwareInterfaceNumber([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIVisaSession  + 5), helpstring("Get the
hardware interface type")]
      HRESULT HardwareInterfaceType([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIVisaSession  + 6), helpstring("Get the current
lock state of the resource")]
      HRESULT LockState([out, retval] AccessMode *pVal);
      [propget, helpcontext(HlpCtxIVisaSession  + 7), helpstring("Get the current
state of all settable properties")]
      HRESULT OptionString([out, retval] BSTR *pVal);
      [propget, helpcontext(HlpCtxIVisaSession  + 8), helpstring("Get the ProgID
of the component")]
      HRESULT ProgID([out, retval] BSTR *pVal);
      [propget, helpcontext(HlpCtxIVisaSession  + 9), helpstring("Get the
resource name")]
      HRESULT ResourceName([out, retval] BSTR *pVal);
      [propget, helpcontext(HlpCtxIVisaSession  + 10), helpstring("Get the
session class type")]
      HRESULT SessionType([out, retval] BSTR *pVal);
      [propget, helpcontext(HlpCtxIVisaSession  + 11), helpstring("Get the
manufacturer ID of the component")]
      HRESULT SoftwareManufacturerID([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIVisaSession  + 12), helpstring("Get the
manufacturer name of the component")]
      HRESULT SoftwareManufacturerName([out, retval] BSTR *pVal);
      [propget, helpcontext(HlpCtxIVisaSession  + 13), helpstring("Get/Set the
I/O timeout in milliseconds")]
      HRESULT Timeout([out, retval] long *pVal);
      [propput, helpcontext(HlpCtxIVisaSession  + 13), helpstring("Get/Set the
I/O timeout in milliseconds")]
      HRESULT Timeout([in] long newVal);
      [propget, helpcontext(HlpCtxIVisaSession  + 14), helpstring("Get the last
status from this session")]
```

```
     HRESULT LastStatus([out, retval] HRESULT *pVal);

     [hidden, helpcontext(HlpCtxIVisaSession  + 15), helpstring("Get the state
of a specified property")]
     HRESULT GetAttribute(
        [in] long attribute,
        [out, retval] VARIANTARG *pAttrState);
     [hidden, helpcontext(HlpCtxIVisaSession  + 16), helpstring("Set the state
of a specified property")]
     HRESULT SetAttribute(
        [in] long attribute,
        [in] VARIANTARG attrState);
     [helpcontext(HlpCtxIVisaSession  + 17), helpstring("Establish ownership of
the resource")]
     HRESULT LockRsrc(
        [in, defaultvalue(EXCLUSIVE_LOCK)] AccessMode type,
        [in, defaultvalue(2000)] long lockTimeout,
        [in, defaultvalue("")] BSTR requestedKey,
        [out, retval] BSTR *pAccessKey);
     [helpcontext(HlpCtxIVisaSession  + 18), helpstring("Relinquish ownership of
the resource")]
     HRESULT UnlockRsrc();
     [helpcontext(HlpCtxIVisaSession  + 19), helpstring("Initialize a session to
the specified resource name")]
     HRESULT Init(
        [in] BSTR resourceName,
        [in, defaultvalue(NO_LOCK)] AccessMode mode,
        [in, defaultvalue(2000)] long initTimeout,
        [in, defaultvalue("")] BSTR optionString);
     [helpcontext(HlpCtxIVisaSession  + 20), helpstring("Close the session")]
     HRESULT Close();
  };

  //-----------------------------------------------------------------------
  //  IBaseMessage
  //-----------------------------------------------------------------------
  [
     object,
     oleautomation,
     helpstring("IBaseMessage - do not use directly"),
     uuid(db8cbf04-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxIBaseMessage + 49),
     pointer_default(unique),
     hidden
  ]
  interface IBaseMessage : IVisaSession
  {
     [propget, helpcontext(HlpCtxIBaseMessage  + 1), helpstring("Get/Set which
I/O protocol to use")]
     HRESULT IOProtocol([out, retval] IOProtocol *pVal);
     [propput, helpcontext(HlpCtxIBaseMessage  + 1), helpstring("Get/Set which
I/O protocol to use")]
     HRESULT IOProtocol([in] IOProtocol newVal);
     [propget, helpcontext(HlpCtxIBaseMessage  + 2), helpstring("Get/Set whether
to assert END on Write")]
     HRESULT SendEndEnabled([out, retval] VARIANT_BOOL *pVal);
     [propput, helpcontext(HlpCtxIBaseMessage  + 2), helpstring("Get/Set whether
to assert END on Write")]
     HRESULT SendEndEnabled([in] VARIANT_BOOL newVal);
     [propget, helpcontext(HlpCtxIBaseMessage  + 3), helpstring("Get/Set the
termination character")]
     HRESULT TerminationCharacter([out, retval] BYTE *pVal);
     [propput, helpcontext(HlpCtxIBaseMessage  + 3), helpstring("Get/Set the
termination character")]
     HRESULT TerminationCharacter([in] BYTE newVal);
     [propget, helpcontext(HlpCtxIBaseMessage  + 4), helpstring("Get/Set whether
to use the termination character on Read")]
     HRESULT TerminationCharacterEnabled([out, retval] VARIANT_BOOL *pVal);
```

```
     [propput, helpcontext(HlpCtxIBaseMessage  + 4), helpstring("Get/Set whether
to use the termination character on Read")]
     HRESULT TerminationCharacterEnabled([in] VARIANT_BOOL newVal);

     [helpcontext(HlpCtxIBaseMessage  + 5), helpstring("Assert a trigger")]
     HRESULT AssertTrigger(
        [in, defaultvalue(TRIG_PROT_DEFAULT)] TriggerProtocol protocol);
     [helpcontext(HlpCtxIBaseMessage  + 6), helpstring("Clear the device")]
     HRESULT Clear();
     [helpcontext(HlpCtxIBaseMessage  + 7), helpstring("Read the status byte")]
     HRESULT ReadSTB(
        [out, retval] short *pStatusByte);
  };

  //-----------------------------------------------------------------------
  //  IMessage
  //-----------------------------------------------------------------------
  [
     object,
     oleautomation,
     helpstring("Message Based Interface"),
     uuid(db8cbf05-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxIMessage + 49),
     pointer_default(unique)
  ]
  interface IMessage : IBaseMessage
  {
     [helpcontext(HlpCtxIMessage  + 1), helpstring("Read the specified number of
bytes")]
     HRESULT Read(
        [in] long count,
        [out, retval] SAFEARRAY(BYTE) *pBuffer);
     [helpcontext(HlpCtxIMessage  + 2), helpstring("Read the specified number of
bytes as a string")]
     HRESULT ReadString(
        [in] long count,
        [out, retval] BSTR *pBuffer);
     [helpcontext(HlpCtxIMessage  + 3), helpstring("Write the specified data")]
     HRESULT Write(
        [in] SAFEARRAY(BYTE) *buffer,
        [in] long count,
        [out, retval] long *pRetCount);
     [helpcontext(HlpCtxIMessage  + 4), helpstring("Write the specified
string")]
     HRESULT WriteString(
        [in] BSTR buffer,
        [out, retval] long *pRetCount);
  };

  //-----------------------------------------------------------------------
  //  IAsyncMessage
  //-----------------------------------------------------------------------
  [
     object,
     oleautomation,
     helpstring("Asynchronous Message Based Interface"),
     uuid(db8cbf06-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxIAsyncMessage + 49),
     pointer_default(unique)
  ]
  interface IAsyncMessage : IBaseMessage
  {
     [helpcontext(HlpCtxIAsyncMessage  + 1), helpstring("Read the specified
number of bytes")]
     HRESULT Read(
        [in] long count,
        [out, retval] long *pJobId);
     [helpcontext(HlpCtxIAsyncMessage  + 2), helpstring("Write the specified
```

```
data")]
     HRESULT Write(
        [in] SAFEARRAY(BYTE) *Buffer,
        [in] long count,
        [out, retval] long *pJobId);
     [helpcontext(HlpCtxIAsyncMessage  + 3), helpstring("Write the specified
string")]
     HRESULT WriteString(
        [in] BSTR buffer,
        [out, retval] long *pJobId);
     [helpcontext(HlpCtxIAsyncMessage  + 4), helpstring("Terminate the specified
asynchronous job")]
     HRESULT Terminate([in] long jobId);
   };

   //------------------------------------------------------------------------
   //  IRegister
   //------------------------------------------------------------------------
   [
      object,
      oleautomation,
      helpstring("Register Based Interface"),
      uuid(db8cbf07-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIRegister + 49),
      pointer_default(unique)
   ]
   interface IRegister : IVisaSession
   {
      [propget, helpcontext(HlpCtxIRegister  + 1), helpstring("Get/Set whether
the target format is Big Endian")]
     HRESULT DestinationBigEndian([out, retval] VARIANT_BOOL *pVal);
      [propput, helpcontext(HlpCtxIRegister  + 1), helpstring("Get/Set whether
the target format is Big Endian")]
     HRESULT DestinationBigEndian([in] VARIANT_BOOL newVal);
      [propget, helpcontext(HlpCtxIRegister  + 2), helpstring("Get/Set the target
increment on Move")]
     HRESULT DestinationIncrement([out, retval] long *pVal);
      [propput, helpcontext(HlpCtxIRegister  + 2), helpstring("Get/Set the target
increment on Move")]
     HRESULT DestinationIncrement([in] long newVal);
      [propget, helpcontext(HlpCtxIRegister  + 3), helpstring("Get/Set whether
the source format is Big Endian")]
     HRESULT SourceBigEndian([out, retval] VARIANT_BOOL *pVal);
      [propput, helpcontext(HlpCtxIRegister  + 3), helpstring("Get/Set whether
the source format is Big Endian")]
     HRESULT SourceBigEndian([in] VARIANT_BOOL newVal);
      [propget, helpcontext(HlpCtxIRegister  + 4), helpstring("Get/Set the source
increment on Move")]
     HRESULT SourceIncrement([out, retval] long *pVal);
      [propput, helpcontext(HlpCtxIRegister  + 4), helpstring("Get/Set the source
increment on Move")]
     HRESULT SourceIncrement([in] long newVal);

      [helpcontext(HlpCtxIRegister  + 5), helpstring("Read a value from the
memory location")]
     HRESULT In8(
        [in] short space,
        [in] long offset,
        [out, retval] BYTE *pVal8);
     [helpcontext(HlpCtxIRegister  + 6), helpstring("Read a value from the
memory location")]
     HRESULT In16(
        [in] short space,
        [in] long offset,
        [out, retval] short *pVal16);
     [helpcontext(HlpCtxIRegister  + 7), helpstring("Read a value from the
memory location")]
     HRESULT In32(
```

```
        [in] short space,
        [in] long offset,
        [out, retval] long *pVal32);
     [helpcontext(HlpCtxIRegister  + 8), helpstring("Write a value to the memory
location")]
     HRESULT Out8(
        [in] short space,
        [in] long offset,
        [in] BYTE val8);
     [helpcontext(HlpCtxIRegister  + 9), helpstring("Write a value to the memory
location")]
     HRESULT Out16(
        [in] short space,
        [in] long offset,
        [in] short val16);
     [helpcontext(HlpCtxIRegister  + 10), helpstring("Write a value to the
memory location")]
     HRESULT Out32(
        [in] short space,
        [in] long offset,
        [in] long val32);
     [helpcontext(HlpCtxIRegister  + 11), helpstring("Read data from the memory
location")]
     HRESULT MoveIn8(
        [in] short space,
        [in] long offset,
        [in] long length,
        [out, retval] SAFEARRAY(BYTE) *pBuf8);
     [helpcontext(HlpCtxIRegister  + 12), helpstring("Read data from the memory
location")]
     HRESULT MoveIn16(
        [in] short space,
        [in] long offset,
        [in] long length,
        [out, retval] SAFEARRAY(short) *pBuf16);
     [helpcontext(HlpCtxIRegister  + 13), helpstring("Read data from the memory
location")]
     HRESULT MoveIn32(
        [in] short space,
        [in] long offset,
        [in] long length,
        [out, retval] SAFEARRAY(long) *pBuf32);
     [helpcontext(HlpCtxIRegister  + 14), helpstring("Write data to the memory
location")]
     HRESULT MoveOut8(
        [in] short space,
        [in] long offset,
        [in] long length,
        [in] SAFEARRAY(BYTE) *buf8);
     [helpcontext(HlpCtxIRegister  + 15), helpstring("Write data to the memory
location")]
     HRESULT MoveOut16(
        [in] short space,
        [in] long offset,
        [in] long length,
        [in] SAFEARRAY(short) *buf16);
     [helpcontext(HlpCtxIRegister  + 16), helpstring("Write data to the memory
location")]
     HRESULT MoveOut32(
        [in] short space,
        [in] long offset,
        [in] long length,
        [in] SAFEARRAY(long) *buf32);
     [helpcontext(HlpCtxIRegister  + 17), helpstring("Move data between memory
locations")]
     HRESULT Move(
        [in] short srcSpace,
        [in] long srcOffset,
```

```
        [in] DataWidth srcWidth,
        [in] short destSpace,
        [in] long destOffset,
        [in] DataWidth destWidth,
        [in] long length);
   };


   //-----------------------------------------------------------------------
   //  IRegister64
   //-----------------------------------------------------------------------
   [
      object,
      oleautomation,
      helpstring("Register Based Interface supporting 64-bit integers
(obsolete)"),
      uuid(DB8CBF29-D6D3-11D4-AA51-00A024EE30BD),
      helpcontext(HlpCtxIRegister64 + 49),
      pointer_default(unique),
      hidden
   ]
   interface IRegister64 : IRegister
   {
      [helpcontext(HlpCtxIRegister64  + 1), helpstring("Read a 64-bit integer
value from the memory location")]
      HRESULT In64(
         [in] short space,
         [in] long offset,
         [out, retval] __int64 *pVal8);

      [helpcontext(HlpCtxIRegister64  + 2), helpstring("Write a 64-bit integer
value to the memory location")]
      HRESULT Out64(
         [in] short space,
         [in] long offset,
         [in] __int64 val8);

      [helpcontext(HlpCtxIRegister64  + 3), helpstring("Read 64-bit integer data
from the memory location")]
      HRESULT MoveIn64(
         [in] short space,
         [in] long offset,
         [in] long length,
         [out, retval] SAFEARRAY(__int64) *pBuf8);

      [helpcontext(HlpCtxIRegister64  + 4), helpstring("Write 64-bit integer data
to the memory location")]
      HRESULT MoveOut64(
         [in] short space,
         [in] long offset,
         [in] long length,
         [in] SAFEARRAY(__int64) *buf8);

      [helpcontext(HlpCtxIRegister64  + 5), helpstring("Read a value from the
memory location")]
      HRESULT In8Ex(
         [in] short space,
         [in] __int64 offset,
         [out, retval] BYTE *pVal8);

      [helpcontext(HlpCtxIRegister64  + 6), helpstring("Read a value from the
memory location")]
      HRESULT In16Ex(
         [in] short space,
         [in] __int64 offset,
         [out, retval] short *pVal16);
```

```
     [helpcontext(HlpCtxIRegister64  + 7), helpstring("Read a value from the
memory location")]
     HRESULT In32Ex(
        [in] short space,
        [in] __int64 offset,
        [out, retval] long *pVal32);

     [helpcontext(HlpCtxIRegister64  + 8), helpstring("Read a value from the
memory location")]
     HRESULT In64Ex(
        [in] short space,
        [in] __int64 offset,
        [out, retval] __int64 *pVal8);

     [helpcontext(HlpCtxIRegister64  + 9), helpstring("Write a value to the
memory location")]
     HRESULT Out8Ex(
        [in] short space,
        [in] __int64 offset,
        [in] BYTE val8);

     [helpcontext(HlpCtxIRegister64  + 10), helpstring("Write a value to the
memory location")]
     HRESULT Out16Ex(
        [in] short space,
        [in] __int64 offset,
        [in] short val16);

     [helpcontext(HlpCtxIRegister64  + 11), helpstring("Write a value to the
memory location")]
     HRESULT Out32Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long val32);

     [helpcontext(HlpCtxIRegister64  + 12), helpstring("Write a value to the
memory location")]
     HRESULT Out64Ex(
        [in] short space,
        [in] __int64 offset,
        [in] __int64 val8);

     [helpcontext(HlpCtxIRegister64  + 13), helpstring("Read data from the
memory location")]
     HRESULT MoveIn8Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [out, retval] SAFEARRAY(BYTE) *pBuf8);

     [helpcontext(HlpCtxIRegister64  + 14), helpstring("Read data from the
memory location")]
     HRESULT MoveIn16Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [out, retval] SAFEARRAY(short) *pBuf16);

     [helpcontext(HlpCtxIRegister64  + 15), helpstring("Read data from the
memory location")]
     HRESULT MoveIn32Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [out, retval] SAFEARRAY(long) *pBuf32);

     [helpcontext(HlpCtxIRegister64  + 16), helpstring("Read data from the
memory location")]
```

```
     HRESULT MoveIn64Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [out, retval] SAFEARRAY(__int64) *pBuf8);

     [helpcontext(HlpCtxIRegister64  + 17), helpstring("Write data to the memory
location")]
     HRESULT MoveOut8Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [in] SAFEARRAY(BYTE) *buf8);

     [helpcontext(HlpCtxIRegister64  + 18), helpstring("Write data to the memory
location")]
     HRESULT MoveOut16Ex(
        [in] short space,
        [in] long offset,
        [in] __int64 length,
        [in] SAFEARRAY(short) *buf16);

     [helpcontext(HlpCtxIRegister64  + 19), helpstring("Write data to the memory
location")]
     HRESULT MoveOut32Ex(
        [in] short space,
        [in] long offset,
        [in] __int64 length,
        [in] SAFEARRAY(long) *buf32);

     [helpcontext(HlpCtxIRegister64  + 20), helpstring("Write data to the memory
location")]
     HRESULT MoveOut64Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [in] SAFEARRAY(__int64) *buf8);

     [helpcontext(HlpCtxIRegister64  + 21), helpstring("Move data between memory
locations")]
     HRESULT MoveEx(
        [in] short srcSpace,
        [in] __int64 srcOffset,
        [in] DataWidth srcWidth,
        [in] short destSpace,
        [in] __int64 destOffset,
        [in] DataWidth destWidth,
        [in] long length);
  };

  //-----------------------------------------------------------------------
  //  IRegister64_2
  //-----------------------------------------------------------------------
  [
     object,
     oleautomation,
     helpstring("Register Based Interface 2 supporting 64-bit integers"),
     uuid(DB8CBF2A-D6D3-11D4-AA51-00A024EE30BD),
     helpcontext(HlpCtxIRegister64 + 50),
     pointer_default(unique)
  ]
  interface IRegister64_2 : IRegister
  {
     [helpcontext(HlpCtxIRegister64  + 1), helpstring("Read a 64-bit integer
value from the memory location")]
     HRESULT In64(
        [in] short space,
        [in] long offset,
```

```
        [out, retval] __int64 *pVal8);

     [helpcontext(HlpCtxIRegister64  + 2), helpstring("Write a 64-bit integer
value to the memory location")]
     HRESULT Out64(
        [in] short space,
        [in] long offset,
        [in] __int64 val8);

     [helpcontext(HlpCtxIRegister64  + 3), helpstring("Read 64-bit integer data
from the memory location")]
     HRESULT MoveIn64(
        [in] short space,
        [in] long offset,
        [in] long length,
        [out, retval] SAFEARRAY(__int64) *pBuf8);

     [helpcontext(HlpCtxIRegister64  + 4), helpstring("Write 64-bit integer data
to the memory location")]
     HRESULT MoveOut64(
        [in] short space,
        [in] long offset,
        [in] long length,
        [in] SAFEARRAY(__int64) *buf8);

     [helpcontext(HlpCtxIRegister64  + 5), helpstring("Read a value from the
memory location")]
     HRESULT In8Ex(
        [in] short space,
        [in] __int64 offset,
        [out, retval] BYTE *pVal8);

     [helpcontext(HlpCtxIRegister64  + 6), helpstring("Read a value from the
memory location")]
     HRESULT In16Ex(
        [in] short space,
        [in] __int64 offset,
        [out, retval] short *pVal16);

     [helpcontext(HlpCtxIRegister64  + 7), helpstring("Read a value from the
memory location")]
     HRESULT In32Ex(
        [in] short space,
        [in] __int64 offset,
        [out, retval] long *pVal32);

     [helpcontext(HlpCtxIRegister64  + 8), helpstring("Read a value from the
memory location")]
     HRESULT In64Ex(
        [in] short space,
        [in] __int64 offset,
        [out, retval] __int64 *pVal8);

     [helpcontext(HlpCtxIRegister64  + 9), helpstring("Write a value to the
memory location")]
     HRESULT Out8Ex(
        [in] short space,
        [in] __int64 offset,
        [in] BYTE val8);

     [helpcontext(HlpCtxIRegister64  + 10), helpstring("Write a value to the
memory location")]
     HRESULT Out16Ex(
        [in] short space,
        [in] __int64 offset,
        [in] short val16);

     [helpcontext(HlpCtxIRegister64  + 11), helpstring("Write a value to the
```

```
memory location")]
     HRESULT Out32Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long val32);

     [helpcontext(HlpCtxIRegister64  + 12), helpstring("Write a value to the
memory location")]
     HRESULT Out64Ex(
        [in] short space,
        [in] __int64 offset,
        [in] __int64 val8);

     [helpcontext(HlpCtxIRegister64  + 13), helpstring("Read data from the
memory location")]
     HRESULT MoveIn8Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [out, retval] SAFEARRAY(BYTE) *pBuf8);

     [helpcontext(HlpCtxIRegister64  + 14), helpstring("Read data from the
memory location")]
     HRESULT MoveIn16Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [out, retval] SAFEARRAY(short) *pBuf16);

     [helpcontext(HlpCtxIRegister64  + 15), helpstring("Read data from the
memory location")]
     HRESULT MoveIn32Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [out, retval] SAFEARRAY(long) *pBuf32);

     [helpcontext(HlpCtxIRegister64  + 16), helpstring("Read data from the
memory location")]
     HRESULT MoveIn64Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [out, retval] SAFEARRAY(__int64) *pBuf8);

     [helpcontext(HlpCtxIRegister64  + 17), helpstring("Write data to the memory
location")]
     HRESULT MoveOut8Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [in] SAFEARRAY(BYTE) *buf8);

     [helpcontext(HlpCtxIRegister64  + 22), helpstring("Write data to the memory
location")]
     HRESULT MoveOut16Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [in] SAFEARRAY(short) *buf16);

     [helpcontext(HlpCtxIRegister64  + 23), helpstring("Write data to the memory
location")]
     HRESULT MoveOut32Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [in] SAFEARRAY(long) *buf32);
```

```
     [helpcontext(HlpCtxIRegister64  + 20), helpstring("Write data to the memory
location")]
     HRESULT MoveOut64Ex(
        [in] short space,
        [in] __int64 offset,
        [in] long length,
        [in] SAFEARRAY(__int64) *buf8);

     [helpcontext(HlpCtxIRegister64  + 21), helpstring("Move data between memory
locations")]
     HRESULT MoveEx(
        [in] short srcSpace,
        [in] __int64 srcOffset,
        [in] DataWidth srcWidth,
        [in] short destSpace,
        [in] __int64 destOffset,
        [in] DataWidth destWidth,
        [in] long length);
  };

  //-----------------------------------------------------------------------
  //  ISharedRegister
  //-----------------------------------------------------------------------
  [
     object,
     oleautomation,
     helpstring("Shared Memory Interface"),
     uuid(db8cbf08-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxISharedRegister + 49),
     pointer_default(unique)
  ]
  interface ISharedRegister : IVisaSession
  {
     [helpcontext(HlpCtxISharedRegister  + 1), helpstring("Allocate memory")]
     HRESULT AllocateMemory(
        [in] long size,
        [out, retval] long *pOffset);
     [helpcontext(HlpCtxISharedRegister  + 2), helpstring("Free memory")]
     HRESULT FreeMemory(
        [in] long offset);
  };


  //-----------------------------------------------------------------------
  //  ISharedRegister64
  //-----------------------------------------------------------------------
  [
     object,
     oleautomation,
     helpstring("Shared Memory Interface supporting 64-bit integers"),
     uuid(DB8CBF26-D6D3-11D4-AA51-00A024EE30BD),
     helpcontext(HlpCtxISharedRegister64 + 49),
     pointer_default(unique)
  ]
  interface ISharedRegister64 : ISharedRegister
  {
     [helpcontext(HlpCtxISharedRegister64 + 1), helpstring("Allocate memory")]
     HRESULT AllocateMemoryEx(
        [in] long size,
        [out, retval] __int64 *pOffset);

     [helpcontext(HlpCtxISharedRegister64 + 2), helpstring("Free memory")]
     HRESULT FreeMemoryEx(
         [in] __int64 offset);
  };

//=========================================================================
```

```
//  BUS Specific Property Interfaces
//============================================================================

   //--------------------------------------------------------------------------
   //  IGpib
   //--------------------------------------------------------------------------
   [
      object,
      oleautomation,
      helpstring("GPIB Interface"),
      uuid(db8cbf09-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIGpib + 49),
      pointer_default(unique)
   ]
   interface IGpib : IVisaSession
   {
      [propget, helpcontext(HlpCtxIGpib  + 1), helpstring("Get the primary
address")]
      HRESULT PrimaryAddress([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIGpib  + 2), helpstring("Get the REN line
state")]
      HRESULT RENState([out, retval] LineState *pVal);
      [propget, helpcontext(HlpCtxIGpib  + 3), helpstring("Get/Set whether to
repeat address")]
      HRESULT RepeatAddressingEnabled([out, retval] VARIANT_BOOL *pVal);
      [propput, helpcontext(HlpCtxIGpib  + 3), helpstring("Get/Set whether to
repeat address")]
      HRESULT RepeatAddressingEnabled([in] VARIANT_BOOL newVal);
      [propget, helpcontext(HlpCtxIGpib  + 4), helpstring("Get the secondary
address")]
      HRESULT SecondaryAddress([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIGpib  + 5), helpstring("Get/Set whether to
unaddress")]
      HRESULT UnaddressingEnabled([out, retval] VARIANT_BOOL *pVal);
      [propput, helpcontext(HlpCtxIGpib  + 5), helpstring("Get/Set whether to
unaddress")]
      HRESULT UnaddressingEnabled([in] VARIANT_BOOL newVal);

      [helpcontext(HlpCtxIGpib  + 6), helpstring("Control the REN line
(remote/local) state")]
      HRESULT ControlREN(
         [in] RENControlConst mode);
   };

   //--------------------------------------------------------------------------
   //  IGpibIntfc
   //--------------------------------------------------------------------------
   [
      object,
      oleautomation,
      helpstring("Board-level GPIB Interface"),
      uuid(db8cbf0a-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIGpibIntfc + 49),
      pointer_default(unique)
   ]
   interface IGpibIntfc : IVisaSession
   {
      [propget, helpcontext(HlpCtxIGpibIntfc  + 1), helpstring("Get the
controller addressing state")]
      HRESULT AddressingState([out, retval] GPIBAddressState *pVal);
      [propget, helpcontext(HlpCtxIGpibIntfc  + 2), helpstring("Get the ATN line
state")]
      HRESULT ATNState([out, retval] LineState *pVal);
      [propget, helpcontext(HlpCtxIGpibIntfc  + 3), helpstring("Get/Set the
status byte")]
      HRESULT DevStatusByte([out, retval] BYTE *pVal);
      [propput, helpcontext(HlpCtxIGpibIntfc  + 3), helpstring("Get/Set the
status byte")]
```

```
     HRESULT DevStatusByte([in] BYTE newVal);
     [propget, helpcontext(HlpCtxIGpibIntfc  + 4), helpstring("Get the
controller CIC state")]
     HRESULT CICState([out, retval] VARIANT_BOOL *pVal);
     [propget, helpcontext(HlpCtxIGpibIntfc  + 5), helpstring("Get/Set the HS-
488 cable length")]
     HRESULT HS488CBLLength([out, retval] short *pVal);
     [propput, helpcontext(HlpCtxIGpibIntfc  + 5), helpstring("Get/Set the HS-
488 cable length")]
     HRESULT HS488CBLLength([in] short newVal);
     [propget, helpcontext(HlpCtxIGpibIntfc  + 6), helpstring("Get the NDAC line
state")]
     HRESULT NDACState([out, retval] LineState *pVal);
     [propget, helpcontext(HlpCtxIGpibIntfc  + 7), helpstring("Get/Set the
primary address")]
     HRESULT PrimaryAddress([out, retval] short *pVal);
     [propput, helpcontext(HlpCtxIGpibIntfc  + 7), helpstring("Get/Set the
primary address")]
     HRESULT PrimaryAddress([in] short newVal);
     [propget, helpcontext(HlpCtxIGpibIntfc  + 8), helpstring("Get the REN line
state")]
     HRESULT RENState([out, retval] LineState *pVal);
     [propget, helpcontext(HlpCtxIGpibIntfc  + 9), helpstring("Get/Set the
secondary address")]
     HRESULT SecondaryAddress([out, retval] short *pVal);
     [propput, helpcontext(HlpCtxIGpibIntfc  + 9), helpstring("Get/Set the
secondary address")]
     HRESULT SecondaryAddress([in] short newVal);
     [propget, helpcontext(HlpCtxIGpibIntfc  + 10), helpstring("Get the SRQ line
state")]
     HRESULT SRQState([out, retval] LineState *pVal);
     [propget, helpcontext(HlpCtxIGpibIntfc  + 11), helpstring("Get/Set the
system controller state")]
     HRESULT SysControlState([out, retval] VARIANT_BOOL *pVal);
     [propput, helpcontext(HlpCtxIGpibIntfc  + 11), helpstring("Get/Set the
system controller state")]
     HRESULT SysControlState([in] VARIANT_BOOL newVal);

     [helpcontext(HlpCtxIGpibIntfc  + 12), helpstring("Write GPIB command bytes
on the bus")]
     HRESULT Command(
        [in] SAFEARRAY(BYTE) *buffer,
        [in] long count,
        [out, retval] long *pRetCount);
     [helpcontext(HlpCtxIGpibIntfc  + 13), helpstring("Control the ATN line
state")]
     HRESULT ControlATN(
        [in] ATNControlConst mode);
     [helpcontext(HlpCtxIGpibIntfc  + 14), helpstring("Control the REN line
(remote/local) state")]
     HRESULT ControlREN(
        [in] RENControlConst mode);
     [helpcontext(HlpCtxIGpibIntfc  + 15), helpstring("Pass control to the
specified device")]
     HRESULT PassControl(
        [in] short primAddr,
        [in, defaultvalue(-1)] short secAddr);
     [helpcontext(HlpCtxIGpibIntfc  + 16), helpstring("Pulse the IFC line")]
     HRESULT SendIFC();
  };

  //-----------------------------------------------------------------------
  //  IGpibIntfcMessage
  //-----------------------------------------------------------------------
  [
     object,
     oleautomation,
     helpstring("Board-level GPIB Message Based Interface"),
```

```
     uuid(db8cbf0b-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxIGpibIntfcMessage + 49),
     pointer_default(unique)
  ]
  interface IGpibIntfcMessage : IVisaSession
  {
     [propget, helpcontext(HlpCtxIGpibIntfcMessage  + 1), helpstring("Get/Set
whether to assert END on Write")]
     HRESULT SendEndEnabled([out, retval] VARIANT_BOOL *pVal);
     [propput, helpcontext(HlpCtxIGpibIntfcMessage  + 1), helpstring("Get/Set
whether to assert END on Write")]
     HRESULT SendEndEnabled([in] VARIANT_BOOL newVal);
     [propget, helpcontext(HlpCtxIGpibIntfcMessage  + 2), helpstring("Get/Set
the termination character")]
     HRESULT TerminationCharacter([out, retval] BYTE *pVal);
     [propput, helpcontext(HlpCtxIGpibIntfcMessage  + 2), helpstring("Get/Set
the termination character")]
     HRESULT TerminationCharacter([in] BYTE newVal);
     [propget, helpcontext(HlpCtxIGpibIntfcMessage  + 3), helpstring("Get/Set
whether to use the termination character on Read")]
     HRESULT TerminationCharacterEnabled([out, retval] VARIANT_BOOL *pVal);
     [propput, helpcontext(HlpCtxIGpibIntfcMessage  + 3), helpstring("Get/Set
whether to use the termination character on Read")]
     HRESULT TerminationCharacterEnabled([in] VARIANT_BOOL newVal);

     [helpcontext(HlpCtxIGpibIntfcMessage  + 4), helpstring("Assert a trigger")]
     HRESULT AssertTrigger(
        [in, defaultvalue(TRIG_PROT_DEFAULT)] TriggerProtocol protocol);
     [helpcontext(HlpCtxIGpibIntfcMessage  + 5), helpstring("Read the specified
number of bytes")]
     HRESULT Read(
        [in] long count,
        [out, retval] SAFEARRAY(BYTE) *pBuffer);
     [helpcontext(HlpCtxIGpibIntfcMessage  + 6), helpstring("Read the specified
number of bytes as a string")]
     HRESULT ReadString(
        [in] long count,
        [out, retval] BSTR *pBuffer);
     [helpcontext(HlpCtxIGpibIntfcMessage  + 7), helpstring("Write the specified
data")]
     HRESULT Write(
        [in] SAFEARRAY(BYTE) *buffer,
        [in] long count,
        [out, retval] long *pRetCount);
     [helpcontext(HlpCtxIGpibIntfcMessage  + 8), helpstring("Write the specified
string")]
     HRESULT WriteString(
        [in] BSTR buffer,
        [out, retval] long *pRetCount);
  };

  //---------------------------------------------------------------------------
  //  ISerial
  //---------------------------------------------------------------------------
  [
     object,
     oleautomation,
     helpstring("Serial Interface"),
     uuid(db8cbf0c-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxISerial + 49),
     pointer_default(unique)
  ]
  interface ISerial : IVisaSession
  {
     [propget, helpcontext(HlpCtxISerial  + 1), helpstring("Get the number of
bytes available")]
     HRESULT BytesAvailable([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxISerial  + 2), helpstring("Get/Set the baud
```

```
rate")]
     HRESULT BaudRate([out, retval] long *pVal);
     [propput, helpcontext(HlpCtxISerial  + 2), helpstring("Get/Set the baud
rate")]
     HRESULT BaudRate([in] long newVal);
     [propget, helpcontext(HlpCtxISerial  + 3), helpstring("Get/Set the number
of data bits")]
     HRESULT DataBits([out, retval] short *pVal);
     [propput, helpcontext(HlpCtxISerial  + 3), helpstring("Get/Set the number
of data bits")]
     HRESULT DataBits([in] short newVal);
     [propget, helpcontext(HlpCtxISerial  + 4), helpstring("Get the CTS line
state")]
     HRESULT ClearToSendState([out, retval] LineState *pVal);
     [propget, helpcontext(HlpCtxISerial  + 5), helpstring("Get the DCD line
state")]
     HRESULT DataCarrierDetectState([out, retval] LineState *pVal);
     [propget, helpcontext(HlpCtxISerial  + 6), helpstring("Get the DSR line
state")]
     HRESULT DataSetReadyState([out, retval] LineState *pVal);
     [propget, helpcontext(HlpCtxISerial  + 7), helpstring("Get/Set the DTR line
state")]
     HRESULT DataTerminalReadyState([out, retval] LineState *pVal);
     [propput, helpcontext(HlpCtxISerial  + 7), helpstring("Get/Set the DTR line
state")]
     HRESULT DataTerminalReadyState([in] LineState newVal);
     [propget, helpcontext(HlpCtxISerial  + 8), helpstring("Get/Set the input
end mode")]
     HRESULT EndIn([out, retval] SerialEndConst *pVal);
     [propput, helpcontext(HlpCtxISerial  + 8), helpstring("Get/Set the input
end mode")]
     HRESULT EndIn([in] SerialEndConst newVal);
     [propget, helpcontext(HlpCtxISerial  + 9), helpstring("Get/Set the output
end mode")]
     HRESULT EndOut([out, retval] SerialEndConst *pVal);
     [propput, helpcontext(HlpCtxISerial  + 9), helpstring("Get/Set the output
end mode")]
     HRESULT EndOut([in] SerialEndConst newVal);
     [propget, helpcontext(HlpCtxISerial  + 10), helpstring("Get/Set the flow
control")]
     HRESULT FlowControl([out, retval] SerialFlowControl *pVal);
     [propput, helpcontext(HlpCtxISerial  + 10), helpstring("Get/Set the flow
control")]
     HRESULT FlowControl([in] SerialFlowControl newVal);
     [propget, helpcontext(HlpCtxISerial  + 11), helpstring("Get/Set the
parity")]
     HRESULT Parity([out, retval] SerialParity *pVal);
     [propput, helpcontext(HlpCtxISerial  + 11), helpstring("Get/Set the
parity")]
     HRESULT Parity([in] SerialParity newVal);
     [propget, helpcontext(HlpCtxISerial  + 12), helpstring("Get the RI line
state")]
     HRESULT RingIndicatorState([out, retval] LineState *pVal);
     [propget, helpcontext(HlpCtxISerial  + 13), helpstring("Get/Set the RTS
line state")]
     HRESULT RequestToSendState([out, retval] LineState *pVal);
     [propput, helpcontext(HlpCtxISerial  + 13), helpstring("Get/Set the RTS
line state")]
     HRESULT RequestToSendState([in] LineState newVal);
     [propget, helpcontext(HlpCtxISerial  + 14), helpstring("Get/Set the number
of stop bits")]
     HRESULT StopBits([out, retval] SerialStopBits *pVal);
     [propput, helpcontext(HlpCtxISerial  + 14), helpstring("Get/Set the number
of stop bits")]
     HRESULT StopBits([in] SerialStopBits newVal);
     [propget, helpcontext(HlpCtxISerial  + 15), helpstring("Get/Set the error
replacement character")]
     HRESULT ReplacementCharacter([out, retval] BYTE *pVal);
```

```
     [propput, helpcontext(HlpCtxISerial  + 15), helpstring("Get/Set the error
replacement character")]
     HRESULT ReplacementCharacter([in] BYTE newVal);
     [propget, helpcontext(HlpCtxISerial  + 16), helpstring("Get/Set the XON
character")]
     HRESULT XONCharacter([out, retval] BYTE *pVal);
     [propput, helpcontext(HlpCtxISerial  + 16), helpstring("Get/Set the XON
character")]
     HRESULT XONCharacter([in] BYTE newVal);
     [propget, helpcontext(HlpCtxISerial  + 17), helpstring("Get/Set the XOFF
character")]
     HRESULT XOFFCharacter([out, retval] BYTE *pVal);
     [propput, helpcontext(HlpCtxISerial  + 17), helpstring("Get/Set the XOFF
character")]
     HRESULT XOFFCharacter([in] BYTE newVal);

     [helpcontext(HlpCtxISerial  + 18), helpstring("Set the serial receive or
transmit buffer size")]
     HRESULT SetBufferSize(
        [in] BufferMask mask,
        [in] long size);
     [helpcontext(HlpCtxISerial  + 19), helpstring("Flush the specified serial
buffer")]
     HRESULT Flush(
        [in, defaultvalue(IO_IN_AND_OUT_BUFS)] BufferMask mask,
        [in, defaultvalue(FALSE)] VARIANT_BOOL discard);
  };

  //-----------------------------------------------------------------------
  //  ITcpipInstr
  //-----------------------------------------------------------------------
  [
     object,
     oleautomation,
     helpstring("TCP/IP Instrument Interface"),
     uuid(db8cbf0d-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxITcpipInstr + 49),
     pointer_default(unique)
  ]
  interface ITcpipInstr : IVisaSession
  {
     [propget, helpcontext(HlpCtxITcpipInstr  + 1), helpstring("Get the TCP/IP
address")]
     HRESULT Address([out, retval] BSTR *pVal);
     [propget, helpcontext(HlpCtxITcpipInstr  + 2), helpstring("Get the TCP/IP
hostname")]
     HRESULT HostName([out, retval] BSTR *pVal);
     [propget, helpcontext(HlpCtxITcpipInstr  + 3), helpstring("Get the LAN
device name")]
     HRESULT DeviceName([out, retval] BSTR *pVal);
  };

  //-----------------------------------------------------------------------
  //  ITcpipSocket
  //-----------------------------------------------------------------------
  [
     object,
     oleautomation,
     helpstring("TCP/IP Socket Interface"),
     uuid(db8cbf0e-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxITcpipSocket + 49),
     pointer_default(unique)
  ]
  interface ITcpipSocket : IVisaSession
  {
     [propget, helpcontext(HlpCtxITcpipSocket  + 1), helpstring("Get the TCP/IP
address")]
     HRESULT Address([out, retval] BSTR *pVal);
```

```
     [propget, helpcontext(HlpCtxITcpipSocket  + 2), helpstring("Get the TCP/IP
hostname")]
     HRESULT HostName([out, retval] BSTR *pVal);
     [propget, helpcontext(HlpCtxITcpipSocket  + 3), helpstring("Get/Set whether
to send keep-alive packets")]
     HRESULT KeepAlive([out, retval] VARIANT_BOOL *pVal);
     [propput, helpcontext(HlpCtxITcpipSocket  + 3), helpstring("Get/Set whether
to send keep-alive packets")]
     HRESULT KeepAlive([in] VARIANT_BOOL newVal);
     [propget, helpcontext(HlpCtxITcpipSocket  + 4), helpstring("Get/Set whether
to use the Nagle algorithm")]
     HRESULT NoDelay([out, retval] VARIANT_BOOL *pVal);
     [propput, helpcontext(HlpCtxITcpipSocket  + 4), helpstring("Get/Set whether
to use the Nagle algorithm")]
     HRESULT NoDelay([in] VARIANT_BOOL newVal);
     [propget, helpcontext(HlpCtxITcpipSocket  + 5), helpstring("Get the TCP/IP
port")]
     HRESULT Port([out, retval] short *pVal);

     [helpcontext(HlpCtxITcpipSocket  + 6), helpstring("Set the socket receive
or transmit buffer size")]
     HRESULT SetBufferSize(
        [in] BufferMask mask,
        [in] long size);
     [helpcontext(HlpCtxITcpipSocket  + 7), helpstring("Flush the specified
socket buffer")]
     HRESULT Flush(
        [in, defaultvalue(IO_IN_AND_OUT_BUFS)] BufferMask mask,
        [in, defaultvalue(FALSE)] VARIANT_BOOL discard);
  };

  //-----------------------------------------------------------------------
  //  IUsb
  //-----------------------------------------------------------------------
  [
     object,
     oleautomation,
     helpstring("USB Interface"),
     uuid(db8cbf24-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxIUsb + 49),
     pointer_default(unique)
  ]
  interface IUsb : IVisaSession
  {
     [propget, helpcontext(HlpCtxIUsb  + 1), helpstring("Get the manufacturer
ID")]
     HRESULT ManufacturerID([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIUsb  + 2), helpstring("Get the manufacturer
name")]
     HRESULT ManufacturerName([out, retval] BSTR *pVal);
     [propget, helpcontext(HlpCtxIUsb  + 3), helpstring("Get the model code")]
     HRESULT ModelCode([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIUsb  + 4), helpstring("Get the model name")]
     HRESULT ModelName([out, retval] BSTR *pVal);
     [propget, helpcontext(HlpCtxIUsb  + 5), helpstring("Get 488.2 Compliance")]
     HRESULT Is4882Compliant([out, retval] VARIANT_BOOL *pVal);
     [propget, helpcontext(HlpCtxIUsb  + 6), helpstring("Get the USB Serial
Number")]
     HRESULT UsbSerialNumber([out, retval] BSTR *pVal);
     [propget, helpcontext(HlpCtxIUsb  + 7), helpstring("Get the USB Interface
Number")]
     HRESULT UsbInterfaceNumber([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIUsb  + 8), helpstring("Get the USB Protocol")]
     HRESULT UsbProtocol([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIUsb  + 9), helpstring("Get/Set the Maximum
Interrupt Size")]
     HRESULT MaximumInterruptSize([out, retval] short *pVal);
     [propput, helpcontext(HlpCtxIUsb  + 9), helpstring("Get/Set the Maximum
```

```
Interrupt Size")]
     HRESULT MaximumInterruptSize([in] short newVal);

     [helpcontext(HlpCtxIUsb  + 10), helpstring("Control the REN line
(remote/local) state")]
     HRESULT ControlREN(
        [in] RENControlConst mode);
     [helpcontext(HlpCtxIUsb  + 11), helpstring("Send Data to the USB Control
Port")]
     HRESULT ControlOut(
        [in] short bmRequestType,
        [in] short bRequest,
        [in] short wValue,
        [in] short wIndex,
        [in] short wLength,
        [in] SAFEARRAY(BYTE) *buffer);
     [helpcontext(HlpCtxIUsb  + 12), helpstring("Request Data from the USB
Control Port")]
     HRESULT ControlIn(
        [in] short bmRequestType,
        [in] short bRequest,
        [in] short wValue,
        [in] short wIndex,
        [in] short wLength,
        [out, retval] SAFEARRAY(BYTE) *pBuf);
   };

   //---------------------------------------------------------------------------
   //  IHislipInstr
   //---------------------------------------------------------------------------
[
     object,
     oleautomation,
     helpstring("High Speed LAN Protocol (HiSLIP) Instrument Interface"),
     uuid(DB8CBF27-D6D3-11D4-AA51-00A024EE30BD),
     helpcontext(HlpCtxIHislipInstr + 49),
     pointer_default(unique)
   ]
   interface IHislipInstr : ITcpipInstr
   {
     [propget, helpcontext(HlpCtxIHislipInstr  + 1), helpstring("Get the
negotiated HiSLIP protocol version")]
     HRESULT ProtocolVersion([out, retval] long *pVal);

     [propget, helpcontext(HlpCtxIHislipInstr  + 2), helpstring("Get/Set the
HiSLIP Maximum Message Size in KB (1024 bytes)")]
     HRESULT MaxMessage([out, retval] long *pVal);
     [propput, helpcontext(HlpCtxIHislipInstr  + 2), helpstring("Get/Set the
HiSLIP Maximum Message Size in KB (1024 bytes)")]
     HRESULT MaxMessage([in] long newVal);

     [propget, helpcontext(HlpCtxIHislipInstr  + 3), helpstring("Get/Set the
HiSLIP Operlap Enabled")]
     HRESULT OverlapEnabled([out, retval] VARIANT_BOOL *pVal);
     [propput, helpcontext(HlpCtxIHislipInstr  + 3), helpstring("Get/Set the
HiSLIP Overlap Enabled ")]
     HRESULT OverlapEnabled([in] VARIANT_BOOL newVal);

     [helpcontext(HlpCtxIHislipInstr  + 4), helpstring("Control the REN line
(remote/local) state")]
     HRESULT ControlREN(
         [in] RENControlConst mode);
};

//---------------------------------------------------------------------------
   //  IVxi (obsolete)
   //---------------------------------------------------------------------------
   [
```

```
    object,
    oleautomation,
    helpstring("VXI Interface (obsolete)"),
    uuid(db8cbf0f-d6d3-11d4-aa51-00a024ee30bd),
    helpcontext(HlpCtxIVxi + 49),
    pointer_default(unique),
    hidden
  ]
  interface IVxi : IVisaSession
  {
    [propget, helpcontext(HlpCtxIVxi  + 1), helpstring("Get the commander's
logical address")]
    HRESULT CommanderLA([out, retval] short *pVal);
    [propget, helpcontext(HlpCtxIVxi  + 2), helpstring("Get/Set the target
address modifier")]
    HRESULT DestinationAccessPrivilege([out, retval] VXIMemoryAccessPrivilege
*pVal);
    [propput, helpcontext(HlpCtxIVxi  + 2), helpstring("Get/Set the target
address modifier")]
    HRESULT DestinationAccessPrivilege([in] VXIMemoryAccessPrivilege newVal);
    [propget, helpcontext(HlpCtxIVxi  + 3), helpstring("Get the VXI device
class")]
    HRESULT DeviceClass([out, retval] VXIDevClass *pVal);
    [propget, helpcontext(HlpCtxIVxi  + 4), helpstring("Get/Set the FDC channel
number")]
    HRESULT FastDataChannel([out, retval] short *pVal);
    [propput, helpcontext(HlpCtxIVxi  + 4), helpstring("Get/Set the FDC channel
number")]
    HRESULT FastDataChannel([in] short newVal);
    [propget, helpcontext(HlpCtxIVxi  + 5), helpstring("Get/Set the FDC mode")]
    HRESULT FastDataChannelMode([out, retval] FDCMode *pVal);
    [propput, helpcontext(HlpCtxIVxi  + 5), helpstring("Get/Set the FDC mode")]
    HRESULT FastDataChannelMode([in] FDCMode newVal);
    [propget, helpcontext(HlpCtxIVxi  + 6), helpstring("Get/Set whether to use
an FDC channel pair")]
    HRESULT FastDataChannelUsePair([out, retval] VARIANT_BOOL *pVal);
    [propput, helpcontext(HlpCtxIVxi  + 6), helpstring("Get/Set whether to use
an FDC channel pair")]
    HRESULT FastDataChannelUsePair([in] VARIANT_BOOL newVal);
    [propget, helpcontext(HlpCtxIVxi  + 7), helpstring("Get whether the device
is this controller's servant")]
    HRESULT ImmediateServant([out, retval] VARIANT_BOOL *pVal);
    [propget, helpcontext(HlpCtxIVxi  + 8), helpstring("Get the logical
address")]
    HRESULT LogicalAddress([out, retval] short *pVal);
    [propget, helpcontext(HlpCtxIVxi  + 9), helpstring("Get the mainframe's
logical address")]
    HRESULT MainframeLogicalAddress([out, retval] short *pVal);
    [propget, helpcontext(HlpCtxIVxi  + 10), helpstring("Get the manufacturer
ID")]
    HRESULT ManufacturerID([out, retval] short *pVal);
    [propget, helpcontext(HlpCtxIVxi  + 11), helpstring("Get the manufacturer
name")]
    HRESULT ManufacturerName([out, retval] BSTR *pVal);
    [propget, helpcontext(HlpCtxIVxi  + 12), helpstring("Get the memory base
address")]
    HRESULT MemoryBase([out, retval] long *pVal);
    [propget, helpcontext(HlpCtxIVxi  + 13), helpstring("Get the memory size")]
    HRESULT MemorySize([out, retval] long *pVal);
    [propget, helpcontext(HlpCtxIVxi  + 14), helpstring("Get the memory
space")]
    HRESULT MemorySpace([out, retval] short *pVal);
    [propget, helpcontext(HlpCtxIVxi  + 15), helpstring("Get the model code")]
    HRESULT ModelCode([out, retval] short *pVal);
    [propget, helpcontext(HlpCtxIVxi  + 16), helpstring("Get the model name")]
    HRESULT ModelName([out, retval] BSTR *pVal);
    [propget, helpcontext(HlpCtxIVxi  + 17), helpstring("Get/Set the trigger
ID")]
```

```
     HRESULT TriggerID([out, retval] TriggerLine *pVal);
     [propput, helpcontext(HlpCtxIVxi  + 17), helpstring("Get/Set the trigger
ID")]
     HRESULT TriggerID([in] TriggerLine newVal);
     [propget, helpcontext(HlpCtxIVxi  + 18), helpstring("Get the device's
slot")]
     HRESULT Slot([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIVxi  + 19), helpstring("Get/Set the source
address modifier")]
     HRESULT SourceAccessPrivilege([out, retval] VXIMemoryAccessPrivilege
*pVal);
     [propput, helpcontext(HlpCtxIVxi  + 19), helpstring("Get/Set the source
address modifier")]
     HRESULT SourceAccessPrivilege([in] VXIMemoryAccessPrivilege newVal);
     [propget, helpcontext(HlpCtxIVxi  + 20), helpstring("Get which trigger
lines are supported")]
     HRESULT TriggerSupport([out, retval] long *pVal);

     [helpcontext(HlpCtxIVxi  + 21), helpstring("Assert a trigger")]
     HRESULT AssertTrigger(
        [in, defaultvalue(TRIG_PROT_DEFAULT)] TriggerProtocol protocol);
     [helpcontext(HlpCtxIVxi  + 22), helpstring("Send a miscellaneous VXI
command or query")]
     HRESULT CommandQuery(
        [in] VXICommandQuery mode,
        [in] long cmd,
        [out, retval] long *pResponse);
  };

  //---------------------------------------------------------------------------
  //  IVxi3
  //---------------------------------------------------------------------------
  [
     object,
     oleautomation,
     helpstring("VXI Interface"),
     uuid(db8cbf22-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxIVxi3 + 49),
     pointer_default(unique)
  ]
  interface IVxi3 : IVxi
  {
     [propget, helpcontext(HlpCtxIVxi3  + 1), helpstring("Get 488.2
Compliance")]
     HRESULT Is4882Compliant([out, retval] VARIANT_BOOL *pVal);
  };

  //---------------------------------------------------------------------------
  //  IVxiMemacc
  //---------------------------------------------------------------------------
  [
     object,
     oleautomation,
     helpstring("VXI Memory Access Interface"),
     uuid(db8cbf10-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxIVxiMemacc + 49),
     pointer_default(unique)
  ]
  interface IVxiMemacc : IRegister
  {
     [propget, helpcontext(HlpCtxIVxiMemacc  + 1), helpstring("Get/Set the
target address modifier")]
     HRESULT DestinationAccessPrivilege([out, retval] VXIMemoryAccessPrivilege
*pVal);
     [propput, helpcontext(HlpCtxIVxiMemacc  + 1), helpstring("Get/Set the
target address modifier")]
     HRESULT DestinationAccessPrivilege([in] VXIMemoryAccessPrivilege newVal);
     [propget, helpcontext(HlpCtxIVxiMemacc  + 2), helpstring("Get/Set the
```

```
source address modifier")]
     HRESULT SourceAccessPrivilege([out, retval] VXIMemoryAccessPrivilege
*pVal);
     [propput, helpcontext(HlpCtxIVxiMemacc  + 2), helpstring("Get/Set the
source address modifier")]
     HRESULT SourceAccessPrivilege([in] VXIMemoryAccessPrivilege newVal);
     [propget, helpcontext(HlpCtxIVxiMemacc  + 3), helpstring("Get the logical
address")]
     HRESULT LogicalAddress([out, retval] short *pVal);
  };

  //------------------------------------------------------------------------
  //  IVxiBackplane
  //------------------------------------------------------------------------
  [
     object,
     oleautomation,
     helpstring("VXI Backplane Interface"),
     uuid(db8cbf11-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxIVxiBackplane + 49),
     pointer_default(unique)
  ]
  interface IVxiBackplane : IVisaSession
  {
     [propget, helpcontext(HlpCtxIVxiBackplane  + 1), helpstring("Get the
mainframe's logical address")]
     HRESULT MainframeLA([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIVxiBackplane  + 2), helpstring("Get/Set the
trigger ID")]
     HRESULT TriggerId([out, retval] TriggerLine *pVal);
     [propput, helpcontext(HlpCtxIVxiBackplane  + 2), helpstring("Get/Set the
trigger ID")]
     HRESULT TriggerId([in] TriggerLine newVal);
     [propget, helpcontext(HlpCtxIVxiBackplane  + 3), helpstring("Get which
trigger lines are asserted")]
     HRESULT TriggerStatus([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIVxiBackplane  + 4), helpstring("Get which
trigger lines are supported")]
     HRESULT TriggerSupport([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIVxiBackplane  + 5), helpstring("Get which
interrupt lines are asserted")]
     HRESULT VxiVmeInterruptStatus([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIVxiBackplane  + 6), helpstring("Get the
SYSFAIL line state")]
     HRESULT VxiVmeSysfailStatus([out, retval] LineState *pVal);

     [helpcontext(HlpCtxIVxiBackplane  + 7), helpstring("Assert the specified
interrupt or signal")]
     HRESULT AssertInterruptSignal(
        [in] AssertInterruptConst mode,
        [in] long statusID);
     [helpcontext(HlpCtxIVxiBackplane  + 8), helpstring("Assert a trigger")]
     HRESULT AssertTrigger(
        [in, defaultvalue(TRIG_PROT_DEFAULT)] TriggerProtocol protocol);
     [helpcontext(HlpCtxIVxiBackplane  + 9), helpstring("Assert or deassert the
specified utility signal")]
     HRESULT AssertUtilSignal(
        [in] AssertUtilityConst line);
     [helpcontext(HlpCtxIVxiBackplane  + 10), helpstring("Map between the
specified trigger lines")]
     HRESULT MapTrigger(
        [in] TriggerLine trigSrc,
        [in] TriggerLine trigDest,
        [in, defaultvalue(0)] short mode);
     [helpcontext(HlpCtxIVxiBackplane  + 11), helpstring("Undo a previous
trigger line mapping")]
     HRESULT UnmapTrigger(
        [in] TriggerLine trigSrc,
```

```
        [in, defaultvalue(TRIG_ALL)] TriggerLine trigDest);
   };

   //-----------------------------------------------------------------------
   //  IPxi
   //-----------------------------------------------------------------------
   [
      object,
      oleautomation,
      helpstring("PXI Interface"),
      uuid(DB8CBF28-D6D3-11D4-AA51-00A024EE30BD),
      helpcontext(HlpCtxIPxi + 49),
      pointer_default(unique)
   ]
   interface IPxi : IVisaSession
   {
      [propget, helpcontext(HlpCtxIPxi  + 1), helpstring("Get the PCI bus
number")]
      HRESULT BusNumber([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIPxi  + 2), helpstring("Get the PCI device
number")]
      HRESULT DevNumber([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIPxi  + 3), helpstring("Get the PCI function
number")]
      HRESULT FuncNumber([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIPxi  + 4), helpstring("Get the slot path")]
      HRESULT SlotPath([out, retval] BSTR *pVal);
      [propget, helpcontext(HlpCtxIPxi  + 5), helpstring("Get the slot number or
special feature connected to local left bus lines")]
      HRESULT SlotLocalBusLeft([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIPxi  + 6), helpstring("Get the slot number or
special feature connected to local right bus lines")]
      HRESULT SlotLocalBusRight([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIPxi  + 7), helpstring("Get the trigger bus
number of this device")]
      HRESULT TriggerBus([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIPxi  + 8), helpstring("Get the PXI star
trigger bus")]
      HRESULT StarTriggerBus([out, retval] short *pVal);
      [propget, helpcontext(HlpCtxIPxi  + 9), helpstring("Get the connected PXI
star line")]
      HRESULT StarTriggerLine([out, retval] short *pVal);

      [propget, helpcontext(HlpCtxIPxi  + 10), helpstring("Get the memory type
used in BAR 0")]
      HRESULT MemTypeBar0([out, retval] PXIMemType *pVal);
      [propget, helpcontext(HlpCtxIPxi  + 11), helpstring("Get the memory type
used in BAR 1")]
      HRESULT MemTypeBar1([out, retval] PXIMemType *pVal);
      [propget, helpcontext(HlpCtxIPxi  + 12), helpstring("Get the memory type
used in BAR 2")]
      HRESULT MemTypeBar2([out, retval] PXIMemType *pVal);
      [propget, helpcontext(HlpCtxIPxi  + 13), helpstring("Get the memory type
used in BAR 3")]
      HRESULT MemTypeBar3([out, retval] PXIMemType *pVal);
      [propget, helpcontext(HlpCtxIPxi  + 14), helpstring("Get the memory type
used in BAR 4")]
      HRESULT MemTypeBar4([out, retval] PXIMemType *pVal);
      [propget, helpcontext(HlpCtxIPxi  + 15), helpstring("Get the memory type
used in BAR 5")]
      HRESULT MemTypeBar5([out, retval] PXIMemType *pVal);

      [propget, helpcontext(HlpCtxIPxi  + 16), helpstring("Get the memory base
address for BAR 0")]
      HRESULT MemBaseBar0([out, retval] long *pVal);
      [propget, helpcontext(HlpCtxIPxi  + 17), helpstring("Get the memory base
address for BAR 1")]
      HRESULT MemBaseBar1([out, retval] long *pVal);
```

```
     [propget, helpcontext(HlpCtxIPxi  + 18), helpstring("Get the memory base
address for BAR 2")]
     HRESULT MemBaseBar2([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 19), helpstring("Get the memory base
address for BAR 3")]
     HRESULT MemBaseBar3([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 20), helpstring("Get the memory base
address for BAR 4")]
     HRESULT MemBaseBar4([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 21), helpstring("Get the memory base
address for BAR 5")]
     HRESULT MemBaseBar5([out, retval] long *pVal);

     [propget, helpcontext(HlpCtxIPxi  + 22), helpstring("Get the memory size
for BAR 0")]
     HRESULT MemSizeBar0([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 23), helpstring("Get the memory size
for BAR 1")]
     HRESULT MemSizeBar1([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 24), helpstring("Get the memory size
for BAR 2")]
     HRESULT MemSizeBar2([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 25), helpstring("Get the memory size
for BAR 3")]
     HRESULT MemSizeBar3([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 26), helpstring("Get the memory size
for BAR 4")]
     HRESULT MemSizeBar4([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 27), helpstring("Get the memory size
for BAR 5")]
     HRESULT MemSizeBar5([out, retval] long *pVal);

     [propget, helpcontext(HlpCtxIPxi  + 28), helpstring("Get the chassis
number")]
     HRESULT ChassisNumber([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 29), helpstring("Get whether the device
is PXI Express")]
     HRESULT IsExpress([out, retval] VARIANT_BOOL *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 30), helpstring("Get the link width
used by the slot")]
     HRESULT SlotLinkWidth([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 31), helpstring("Get the maximum usable
link width")]
     HRESULT MaxLinkWidth([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 32), helpstring("Get the negotiated
link width")]
     HRESULT ActualLinkWidth([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 33), helpstring("Get the differential
star bus number")]
     HRESULT DstarBusNumber([out, retval] short *pVal);
     [propget, helpcontext(HlpCtxIPxi  + 34), helpstring("Get the connected set
of PXI Express differential star bus lines")]
     HRESULT DstarLineSet([out, retval] short *pVal);
  };

//=============================================================================
//  Event Management and Events
//=============================================================================

  //-----------------------------------------------------------------------
  //  IEvent
  //-----------------------------------------------------------------------
  [
     object,
     oleautomation,
     helpstring("VISA Event Interface"),
     uuid(db8cbf12-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxIEvent + 49),
```

```
     pointer_default(unique)
   ]
   interface IEvent : IUnknown
   {
      [propget, helpcontext(HlpCtxIEvent  + 1), helpstring("Get the event type")]
      HRESULT Type([out, retval] EventType *pVal);
      [propget, helpcontext(HlpCtxIEvent  + 2), helpstring("Get the custom event
type number")]
      HRESULT CustomEventTypeNumber([out, retval] long *pVal);

      [hidden, helpcontext(HlpCtxIEvent  + 3), helpstring("Get an attribute of
the event")]
      HRESULT GetAttribute(
         [in] long attribute,
         [out, retval] VARIANTARG *pAttrState);
      [hidden, helpcontext(HlpCtxIEvent  + 4), helpstring("Set an attribute of
the event")]
      HRESULT SetAttribute(
         [in] long attribute,
         [in] VARIANTARG attrState);
      [helpcontext(HlpCtxIEvent  + 5), helpstring("Close the event")]
      HRESULT Close();
   };

   //--------------------------------------------------------------------------
   //  IEventHandler
   //--------------------------------------------------------------------------
   [
      object,
      oleautomation,
      helpstring("User-implemented Event Handler Interface"),
      uuid(db8cbf13-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIEventHandler + 49),
      pointer_default(unique)
   ]
   interface IEventHandler : IUnknown
   {
      [helpcontext(HlpCtxIEventHandler  + 1), helpstring("User-implemented event
handler")]
      HRESULT HandleEvent(
         [in] IEventManager *vi,
         [in] IEvent *event,
         [in] long userHandle);
   };

   //--------------------------------------------------------------------------
   //  IEventManager
   //--------------------------------------------------------------------------
   [
      object,
      oleautomation,
      helpstring("Event Manager Interface"),
      uuid(db8cbf14-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIEventManager + 49),
      pointer_default(unique)
   ]
   interface IEventManager : IVisaSession
   {
      [propget, helpcontext(HlpCtxIEventManager  + 1), helpstring("Get/Set the
queue length")]
      HRESULT MaximumQueueLength([out, retval] long *pVal);
      [propput, helpcontext(HlpCtxIEventManager  + 1), helpstring("Get/Set the
queue length")]
      HRESULT MaximumQueueLength([in] long newVal);

      [helpcontext(HlpCtxIEventManager  + 2), helpstring("Enable the specified
event")]
      HRESULT EnableEvent(
```

```
        [in] EventType type,
        [in] EventMechanism mech,
        [in, defaultvalue(0)] long customEventType);
     [helpcontext(HlpCtxIEventManager  + 3), helpstring("Disable the specified
event")]
     HRESULT DisableEvent(
        [in, defaultvalue(ALL_ENABLED_EVENTS)] EventType type,
        [in, defaultvalue(EVENT_ALL_MECH)] EventMechanism mech,
        [in, defaultvalue(0)] long customEventType);
     [helpcontext(HlpCtxIEventManager  + 4), helpstring("Discard events from the
queue")]
     HRESULT DiscardEvents(
        [in, defaultvalue(ALL_ENABLED_EVENTS)] EventType type,
        [in, defaultvalue(EVENT_ALL_MECH)] EventMechanism mech,
        [in, defaultvalue(0)] long customEventType);
     [helpcontext(HlpCtxIEventManager  + 5), helpstring("Wait for the specified
event")]
     HRESULT WaitOnEvent(
        [in] long waitTimeout,
        [in, defaultvalue(ALL_ENABLED_EVENTS)] EventType type,
        [in, defaultvalue(0)] long customEventType,
        [out, retval] IEvent **pEvent);
     [helpcontext(HlpCtxIEventManager  + 6), helpstring("Install a handler for
event callbacks")]
     HRESULT InstallHandler(
        [in] EventType type,
        [in] IEventHandler *handler,
        [in, defaultvalue(0)] long userHandle,
        [in, defaultvalue(0)] long customEventType);
     [helpcontext(HlpCtxIEventManager  + 7), helpstring("Remove a previously
installed handler")]
     HRESULT UninstallHandler(
        [in] EventType type,
        [in, defaultvalue(0)] long userHandle,
        [in, defaultvalue(0)] long customEventType);
  };

  //-----------------------------------------------------------------------
  //  IEventIOCompletion
  //-----------------------------------------------------------------------
  [
     object,
     oleautomation,
     helpstring("I/O Completion Event Interface"),
     uuid(db8cbf15-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxIEventIOCompletion + 49),
     pointer_default(unique)
  ]
  interface IEventIOCompletion : IEvent
  {
     [propget, helpcontext(HlpCtxIEventIOCompletion  + 1), helpstring("Get the
I/O status code of this transfer")]
     HRESULT IOStatus([out, retval] HRESULT *pVal);
     [propget, helpcontext(HlpCtxIEventIOCompletion  + 2), helpstring("Get the
job ID")]
     HRESULT JobId([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIEventIOCompletion  + 3), helpstring("Get the
number of elements transferred")]
     HRESULT ReturnCount([out, retval] long *pVal);
     [propget, helpcontext(HlpCtxIEventIOCompletion  + 4), helpstring("Get the
read buffer data")]
     HRESULT ReadBuffer([out, retval] SAFEARRAY(BYTE) *pVal);
     [propget, helpcontext(HlpCtxIEventIOCompletion  + 5), helpstring("Get the
read buffer as a string")]
     HRESULT ReadBufferAsString([out, retval] BSTR *pVal);
  };

  //-----------------------------------------------------------------------
```

```
   //  IEventTrigger
   //--------------------------------------------------------------------------
   [
      object,
      oleautomation,
      helpstring("Trigger Event Interface"),
      uuid(db8cbf16-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIEventTrigger + 49),
      pointer_default(unique)
   ]
   interface IEventTrigger : IEvent
   {
      [propget, helpcontext(HlpCtxIEventTrigger  + 1), helpstring("Get the
trigger line on which this event was received")]
      HRESULT TriggerID([out, retval] TriggerLine *pVal);
   };

   //--------------------------------------------------------------------------
   //  IEventVxiSignal
   //--------------------------------------------------------------------------
   [
      object,
      oleautomation,
      helpstring("VXI Signal Event Interface"),
      uuid(db8cbf17-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIEventVxiSignal + 49),
      pointer_default(unique)
   ]
   interface IEventVxiSignal : IEvent
   {
      [propget, helpcontext(HlpCtxIEventVxiSignal  + 1), helpstring("Get the 16-
bit signal Status/ID value")]
      HRESULT SignalStatusID([out, retval] short *pVal);
   };

   //--------------------------------------------------------------------------
   //  IEventVxiVmeInterrupt
   //--------------------------------------------------------------------------
   [
      object,
      oleautomation,
      helpstring("VXI/VME Interrupt Event Interface"),
      uuid(db8cbf18-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIEventVxiVmeInterrupt + 49),
      pointer_default(unique)
   ]
   interface IEventVxiVmeInterrupt : IEvent
   {
      [propget, helpcontext(HlpCtxIEventVxiVmeInterrupt  + 1), helpstring("Get
the 32-bit interrupt Status/ID value")]
      HRESULT InterruptStatusID([out, retval] long *pVal);
      [propget, helpcontext(HlpCtxIEventVxiVmeInterrupt  + 2), helpstring("Get
the interrupt level on which this event was received")]
      HRESULT InterruptLevel([out, retval] short *pVal);
   };

   //--------------------------------------------------------------------------
   //  IEventGpibCIC
   //--------------------------------------------------------------------------
   [
      object,
      oleautomation,
      helpstring("GPIB CIC Event Interface"),
      uuid(db8cbf19-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIEventGpibCIC + 49),
      pointer_default(unique)
   ]
   interface IEventGpibCIC : IEvent
```

```
   {
      [propget, helpcontext(HlpCtxIEventGpibCIC  + 1), helpstring("Get the
controller CIC state")]
      HRESULT CICState([out, retval] VARIANT_BOOL *pVal);
   };

   //--------------------------------------------------------------------------
   //  IEventUsbInterrupt
   //--------------------------------------------------------------------------
   [
      object,
      oleautomation,
      helpstring("USB Interrupt Event Interface"),
      uuid(db8cbf23-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIEventUsbInterrupt + 49),
      pointer_default(unique)
   ]
   interface IEventUsbInterrupt : IEvent
   {
      [propget, helpcontext(HlpCtxIEventUsbInterrupt  + 1), helpstring("Get the
received buffer data")]
      HRESULT DataBuffer([out, retval] SAFEARRAY(BYTE) *pVal);
      [propget, helpcontext(HlpCtxIEventUsbInterrupt  + 2), helpstring("Get the
I/O status code of this transfer")]
      HRESULT IOStatus([out, retval] HRESULT *pVal);
      [propget, helpcontext(HlpCtxIEventUsbInterrupt  + 3), helpstring("Get the
actual number of bytes received")]
      HRESULT InterruptSize([out, retval] short *pVal);
   };

//============================================================================
//  Formatted I/O
//============================================================================

   //--------------------------------------------------------------------------
   //  IFormattedIO488
   //--------------------------------------------------------------------------
   [
      object,
      oleautomation,
      helpstring("IEEE 488.2 Formatted I/O Interface"),
      uuid(db8cbf1a-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIFormattedIO488 + 49),
      pointer_default(unique)
   ]
   interface IFormattedIO488 : IUnknown
   {
      typedef [public, helpcontext(HlpCtxEnumIEEEASCIIType), helpstring("ASCII
Data Types"), v1_enum]
      enum IEEEASCIIType {
         [helpcontext(HlpCtxEnumIEEEASCIIType + 1)] ASCIIType_I2 = 2,
         [helpcontext(HlpCtxEnumIEEEASCIIType + 2)] ASCIIType_I4 = 3,
         [helpcontext(HlpCtxEnumIEEEASCIIType + 3)] ASCIIType_R4 = 4,
         [helpcontext(HlpCtxEnumIEEEASCIIType + 4)] ASCIIType_R8 = 5,
         [helpcontext(HlpCtxEnumIEEEASCIIType + 5)] ASCIIType_BSTR = 8,
         [helpcontext(HlpCtxEnumIEEEASCIIType + 6)] ASCIIType_Any = 12,
         [helpcontext(HlpCtxEnumIEEEASCIIType + 7)] ASCIIType_UI1 = 17
      } IEEEASCIIType;
      typedef [public, helpcontext(HlpCtxEnumIEEEBinaryType), helpstring("Binary
Data Types"), v1_enum]
      enum IEEEBinaryType {
         [helpcontext(HlpCtxEnumIEEEBinaryType + 1)] BinaryType_I2 = 2,
         [helpcontext(HlpCtxEnumIEEEBinaryType + 2)] BinaryType_I4 = 3,
         [helpcontext(HlpCtxEnumIEEEBinaryType + 3)] BinaryType_R4 = 4,
         [helpcontext(HlpCtxEnumIEEEBinaryType + 4)] BinaryType_R8 = 5,
         [helpcontext(HlpCtxEnumIEEEBinaryType + 5)] BinaryType_UI1 = 17
      } IEEEBinaryType;
```

```
     [propget,helpstring("Get/Set the I/O Stream to
use"),helpcontext(HlpCtxIFormattedIO488 + 1)]
     HRESULT IO([out, retval] IMessage **pVal);
     [propputref,helpstring("Get/Set the I/O Stream to
use"),helpcontext(HlpCtxIFormattedIO488 + 1)]
     HRESULT IO([in] IMessage *newVal);
     [propget,helpstring("Get/Set whether the instrument communicates in Big
Endian (IEEE 488.2) format"),helpcontext(HlpCtxIFormattedIO488 + 2)]
     HRESULT InstrumentBigEndian([out, retval] VARIANT_BOOL *pVal);
     [propput,helpstring("Get/Set whether the instrument communicates in Big
Endian (IEEE 488.2) format"),helpcontext(HlpCtxIFormattedIO488 + 2)]
     HRESULT InstrumentBigEndian([in] VARIANT_BOOL newVal);

     [helpstring("Write a string to the I/O Stream and optionally flush the
buffer"),helpcontext(HlpCtxIFormattedIO488 + 3)]
     HRESULT WriteString(
        [in] BSTR data,
        [in, defaultvalue(TRUE)] VARIANT_BOOL flushAndEND);
     [helpstring("Write a single number to the I/O Stream and optionally flush
the buffer"),helpcontext(HlpCtxIFormattedIO488 + 4)]
     HRESULT WriteNumber(
        [in] VARIANT data,
        [in, defaultvalue(ASCIIType_Any)] IEEEASCIIType type,
        [in, defaultvalue(TRUE)] VARIANT_BOOL flushAndEND);
     [helpstring("Write a list of values to the I/O Stream and optionally flush
the buffer"),helpcontext(HlpCtxIFormattedIO488 + 5)]
     HRESULT WriteList(
        [in] VARIANT *data,
        [in, defaultvalue(ASCIIType_Any)] IEEEASCIIType type,
        [in, defaultvalue(",")] BSTR listSeperator,
        [in, defaultvalue(TRUE)] VARIANT_BOOL flushAndEND);
     [helpstring("Write a command followed by an IEEE 488.2 definite-length
binary block terminated with the Stream's termination character to the I/O
Stream"),helpcontext(HlpCtxIFormattedIO488 + 6)]
     HRESULT WriteIEEEBlock(
        [in] BSTR command,
        [in] VARIANT data,
        [in, defaultvalue(TRUE)] VARIANT_BOOL flushAndEND);
     [helpstring("Read the entire contents of the buffer until the termination
character / END signal and return the data as a
string"),helpcontext(HlpCtxIFormattedIO488 + 7)]
     HRESULT ReadString(
        [out, retval] BSTR *pData);
     [helpstring("Read a single number from the I/O Stream and optionally flush
the buffer"),helpcontext(HlpCtxIFormattedIO488 + 8)]
     HRESULT ReadNumber(
        [in, defaultvalue(ASCIIType_Any)] IEEEASCIIType type,
        [in, defaultvalue(TRUE)] VARIANT_BOOL flushToEND,
        [out, retval] VARIANT *pData);
     [helpstring("Read a list of values in ASCII format from the I/O Stream,
convert them to the specified type, and optionally flush the
buffer"),helpcontext(HlpCtxIFormattedIO488 + 9)]
     HRESULT ReadList(
        [in, defaultvalue(ASCIIType_Any)] IEEEASCIIType type,
        [in, defaultvalue(",;")] BSTR listSeperator,
        [out, retval] VARIANT *pData);
     [helpstring("Read a definite-length IEEE block from the I/O Stream and
optionally flush the buffer"),helpcontext(HlpCtxIFormattedIO488 + 10)]
     HRESULT ReadIEEEBlock(
        [in] IEEEBinaryType type,
        [in, defaultvalue(FALSE)] VARIANT_BOOL seekToBlock,
        [in, defaultvalue(TRUE)] VARIANT_BOOL flushToEND,
        [out, retval] VARIANT *pData);
     [helpstring("Flush the Write Buffer and optionally send the END
signal"),helpcontext(HlpCtxIFormattedIO488 + 11)]
     HRESULT FlushWrite(
        [in, defaultvalue(TRUE)] VARIANT_BOOL sendEND);
     [helpstring("Flush the Read Buffer"),helpcontext(HlpCtxIFormattedIO488 +
```

```
12)]
     HRESULT FlushRead();
     [helpstring("Set the formatted I/O read or write buffer
size"),helpcontext(HlpCtxIFormattedIO488 + 13)]
     HRESULT SetBufferSize(
        [in] enum BufferMask mask,
        [in] long size);
  };

//=============================================================================
//  VISA Resource Conflict Manager
//=============================================================================

   //---------------------------------------------------------------------
   //  IVisaConflictTableManager
   //---------------------------------------------------------------------
   [
      object,
      oleautomation,
      helpstring("VISA Resource Conflict Manager Interface"),
      uuid(db8cbf1b-d6d3-11d4-aa51-00a024ee30bd),
      helpcontext(HlpCtxIConflictManager + 49),
      pointer_default(unique),
      hidden
   ]
   interface IVisaConflictTableManager : IUnknown
   {
      typedef [public, helpcontext(HlpCtxEnumConflictHandlerType),
helpstring("GUID Handler Types")]
      enum ConflictHandlerType {
         [helpcontext(HlpCtxEnumConflictHandlerType + 1)] NotChosen,
         [helpcontext(HlpCtxEnumConflictHandlerType + 2)] ChosenByResourceManager,
         [helpcontext(HlpCtxEnumConflictHandlerType + 3)] ChosenByUser
      } ConflictHandlerType;

      [propget,helpstring("Get/Set whether to store just conflicts or all
resources"),helpcontext(HlpCtxIConflictManager + 1)]
      HRESULT StoreConflictsOnly([out, retval] VARIANT_BOOL *pVal);
      [propput,helpstring("Get/Set whether to store just conflicts or all
resources"),helpcontext(HlpCtxIConflictManager + 1)]
      HRESULT StoreConflictsOnly([in] VARIANT_BOOL newVal);
      [propget,helpstring("Get the filename of the conflict
table"),helpcontext(HlpCtxIConflictManager + 2)]
      HRESULT ConflictTableFilename([out, retval] BSTR *pVal);
      [propget,helpstring("Get the number of resource entries in the
table"),helpcontext(HlpCtxIConflictManager + 3)]
      HRESULT NumberOfResources([out, retval] long *pVal);

      [helpstring("Add or update a handler in the
table"),helpcontext(HlpCtxIConflictManager + 4)]
      HRESULT CreateHandler(
         [in] short interfaceType,
         [in] short interfaceNumber,
         [in] BSTR sessionType,
         [in] BSTR vsrmGuid,
         [in] ConflictHandlerType type,
         [in, defaultvalue("")] BSTR miscComments);
      [helpstring("Remove a specific handler from the
table"),helpcontext(HlpCtxIConflictManager + 5)]
      HRESULT DeleteHandler(
         [in] short interfaceType,
         [in] short interfaceNumber,
         [in] BSTR sessionType,
         [in] BSTR vsrmGuid);
      [helpstring("Remove all non-user-specified handlers for a given
GUID"),helpcontext(HlpCtxIConflictManager + 6)]
      HRESULT DeleteHandlerByGUID(
         [in] BSTR vsrmGuid);
```

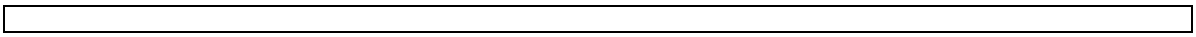
```
     [helpstring("Remove a resource entry from the
table"),helpcontext(HlpCtxIConflictManager + 7)]
     HRESULT DeleteResourceByIndex(
        [in] long tableIndex);
     [helpstring("Find the specified handler for a given
resource"),helpcontext(HlpCtxIConflictManager + 8)]
     HRESULT FindChosenHandler(
        [in] short interfaceType,
        [in] short interfaceNumber,
        [in] BSTR sessionType,
        [in, out] BSTR *pVsrmGuid,
        [in, out] ConflictHandlerType *pType);
     [helpstring("Get the resource information for a given
index"),helpcontext(HlpCtxIConflictManager + 9)]
     HRESULT QueryResource(
        [in] long tableIndex,
        [in, out] short *pInterfaceType,
        [in, out] short *pInterfaceNumber,
        [in, out] BSTR *pSessionType,
        [in, out] short *pNumHandlers);
     [helpstring("Get the handler information for a given
resource"),helpcontext(HlpCtxIConflictManager + 10)]
     HRESULT QueryResourceHandler(
        [in] long tableIndex,
        [in] short handlerIndex,
        [in, out] BSTR *pVsrmGuid,
        [in, out] ConflictHandlerType *pType,
        [in, out] BSTR *pMiscComments);
     [helpstring("Save any changes"),helpcontext(HlpCtxIConflictManager + 11)]
     HRESULT FlushToFile();
   };

//============================================================================
//  CoClasses
//============================================================================

   [
     uuid(db8cbf1c-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxClsResourceManager),
     helpstring("VISA Resource Manager Class")
   ]
   coclass ResourceManager
   {
      [default] interface IResourceManager3;
      interface IResourceManager;
   };

   [
     uuid(db8cbf1d-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxClsFormattedIO488),
     helpstring("IEEE 488.2 Formatted I/O Class")
   ]
   coclass FormattedIO488
   {
      [default] interface IFormattedIO488;
   };

   [
     uuid(db8cbf1f-d6d3-11d4-aa51-00a024ee30bd),
     helpcontext(HlpCtxClsVisaConflictTableManager),
     helpstring("VISA Resource Conflict Manager Class"),
     hidden
   ]
   coclass VisaConflictTableManager
   {
      [default] interface IVisaConflictTableManager;
   };
};
```



8.2. VisaType.idl

```
/*-------------------------------------------------------------------------*/
/* Distributed by VXIplug&play Systems Alliance                            */
/*                                                                         */
/* Do not modify the contents of this file.                                */
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Title   : VISATYPE.IDL                                                  */
/* Date    : 01-14-03                                                      */
/* Purpose : Fundamental VISA data types and macro definitions             */
/*                                                                         */
/*-------------------------------------------------------------------------*/

#ifndef __VISATYPE_IDL_HEADER__
#define __VISATYPE_IDL_HEADER__

#define _VI_ERROR           (-2147483647L-1)  /* 0x80000000 */

/*- Completion and Error Codes ------------------------------------------*/

#define VI_SUCCESS                              (0x00000000L) /* 00000000,
0 */
#define VI_SUCCESS_EVENT_EN                     (0x3FFF0002L) /* 3FFF0002,
1073676290 */
#define VI_SUCCESS_EVENT_DIS                    (0x3FFF0003L) /* 3FFF0003,
1073676291 */
#define VI_SUCCESS_QUEUE_EMPTY                  (0x3FFF0004L) /* 3FFF0004,
1073676292 */
#define VI_SUCCESS_TERM_CHAR                    (0x3FFF0005L) /* 3FFF0005,
1073676293 */
#define VI_SUCCESS_MAX_CNT                      (0x3FFF0006L) /* 3FFF0006,
1073676294 */
#define VI_SUCCESS_DEV_NPRESENT                 (0x3FFF007DL) /* 3FFF007D,
1073676413 */
#define VI_SUCCESS_QUEUE_NEMPTY                 (0x3FFF0080L) /* 3FFF0080,
1073676416 */
#define VI_SUCCESS_TRIG_MAPPED                  (0x3FFF007EL) /* 3FFF007E,
1073676414 */
#define VI_SUCCESS_NCHAIN                       (0x3FFF0098L) /* 3FFF0098,
1073676440 */
#define VI_SUCCESS_NESTED_SHARED                (0x3FFF0099L) /* 3FFF0099,
1073676441 */
#define VI_SUCCESS_NESTED_EXCLUSIVE             (0x3FFF009AL) /* 3FFF009A,
1073676442 */
#define VI_SUCCESS_SYNC                         (0x3FFF009BL) /* 3FFF009B,
1073676443 */
#define VI_WARN_QUEUE_OVERFLOW                  (0x3FFF000CL) /* 3FFF000C,
1073676300 */
#define VI_WARN_CONFIG_NLOADED                  (0x3FFF0077L) /* 3FFF0077,
1073676407 */
#define VI_WARN_NULL_OBJECT                     (0x3FFF0082L) /* 3FFF0082,
1073676418 */
#define VI_WARN_NSUP_ATTR_STATE                 (0x3FFF0084L) /* 3FFF0084,
1073676420 */
#define VI_WARN_UNKNOWN_STATUS                  (0x3FFF0085L) /* 3FFF0085,
1073676421 */
#define VI_WARN_NSUP_BUF                        (0x3FFF0088L) /* 3FFF0088,
1073676424 */
#define VI_WARN_EXT_FUNC_NIMPL                  (0x3FFF00A9L) /* 3FFF00A9,
1073676457 */

#define VI_ERROR_SYSTEM_ERROR       (_VI_ERROR+0x3FFF0000L) /* BFFF0000, -
1073807360 */
#define VI_ERROR_INV_OBJECT         (_VI_ERROR+0x3FFF000EL) /* BFFF000E, -
1073807346 */
```

```
#define VI_ERROR_RSRC_LOCKED        (_VI_ERROR+0x3FFF000FL) /* BFFF000F, -
1073807345 */
#define VI_ERROR_INV_EXPR           (_VI_ERROR+0x3FFF0010L) /* BFFF0010, -
1073807344 */
#define VI_ERROR_RSRC_NFOUND        (_VI_ERROR+0x3FFF0011L) /* BFFF0011, -
1073807343 */
#define VI_ERROR_INV_RSRC_NAME      (_VI_ERROR+0x3FFF0012L) /* BFFF0012, -
1073807342 */
#define VI_ERROR_INV_ACC_MODE       (_VI_ERROR+0x3FFF0013L) /* BFFF0013, -
1073807341 */
#define VI_ERROR_TMO                (_VI_ERROR+0x3FFF0015L) /* BFFF0015, -
1073807339 */
#define VI_ERROR_CLOSING_FAILED     (_VI_ERROR+0x3FFF0016L) /* BFFF0016, -
1073807338 */
#define VI_ERROR_INV_DEGREE         (_VI_ERROR+0x3FFF001BL) /* BFFF001B, -
1073807333 */
#define VI_ERROR_INV_JOB_ID         (_VI_ERROR+0x3FFF001CL) /* BFFF001C, -
1073807332 */
#define VI_ERROR_NSUP_ATTR          (_VI_ERROR+0x3FFF001DL) /* BFFF001D, -
1073807331 */
#define VI_ERROR_NSUP_ATTR_STATE    (_VI_ERROR+0x3FFF001EL) /* BFFF001E, -
1073807330 */
#define VI_ERROR_ATTR_READONLY      (_VI_ERROR+0x3FFF001FL) /* BFFF001F, -
1073807329 */
#define VI_ERROR_INV_LOCK_TYPE      (_VI_ERROR+0x3FFF0020L) /* BFFF0020, -
1073807328 */
#define VI_ERROR_INV_ACCESS_KEY     (_VI_ERROR+0x3FFF0021L) /* BFFF0021, -
1073807327 */
#define VI_ERROR_INV_EVENT          (_VI_ERROR+0x3FFF0026L) /* BFFF0026, -
1073807322 */
#define VI_ERROR_INV_MECH           (_VI_ERROR+0x3FFF0027L) /* BFFF0027, -
1073807321 */
#define VI_ERROR_HNDLR_NINSTALLED   (_VI_ERROR+0x3FFF0028L) /* BFFF0028, -
1073807320 */
#define VI_ERROR_INV_HNDLR_REF      (_VI_ERROR+0x3FFF0029L) /* BFFF0029, -
1073807319 */
#define VI_ERROR_INV_CONTEXT        (_VI_ERROR+0x3FFF002AL) /* BFFF002A, -
1073807318 */
#define VI_ERROR_QUEUE_OVERFLOW     (_VI_ERROR+0x3FFF002DL) /* BFFF002D, -
1073807315 */
#define VI_ERROR_NENABLED           (_VI_ERROR+0x3FFF002FL) /* BFFF002F, -
1073807313 */
#define VI_ERROR_ABORT              (_VI_ERROR+0x3FFF0030L) /* BFFF0030, -
1073807312 */
#define VI_ERROR_RAW_WR_PROT_VIOL   (_VI_ERROR+0x3FFF0034L) /* BFFF0034, -
1073807308 */
#define VI_ERROR_RAW_RD_PROT_VIOL   (_VI_ERROR+0x3FFF0035L) /* BFFF0035, -
1073807307 */
#define VI_ERROR_OUTP_PROT_VIOL     (_VI_ERROR+0x3FFF0036L) /* BFFF0036, -
1073807306 */
#define VI_ERROR_INP_PROT_VIOL      (_VI_ERROR+0x3FFF0037L) /* BFFF0037, -
1073807305 */
#define VI_ERROR_BERR               (_VI_ERROR+0x3FFF0038L) /* BFFF0038, -
1073807304 */
#define VI_ERROR_IN_PROGRESS        (_VI_ERROR+0x3FFF0039L) /* BFFF0039, -
1073807303 */
#define VI_ERROR_INV_SETUP          (_VI_ERROR+0x3FFF003AL) /* BFFF003A, -
1073807302 */
#define VI_ERROR_QUEUE_ERROR        (_VI_ERROR+0x3FFF003BL) /* BFFF003B, -
1073807301 */
#define VI_ERROR_ALLOC              (_VI_ERROR+0x3FFF003CL) /* BFFF003C, -
1073807300 */
#define VI_ERROR_INV_MASK           (_VI_ERROR+0x3FFF003DL) /* BFFF003D, -
1073807299 */
#define VI_ERROR_IO                 (_VI_ERROR+0x3FFF003EL) /* BFFF003E, -
1073807298 */
#define VI_ERROR_INV_FMT            (_VI_ERROR+0x3FFF003FL) /* BFFF003F, -
1073807297 */
```

```
#define VI_ERROR_NSUP_FMT           (_VI_ERROR+0x3FFF0041L) /* BFFF0041, -
1073807295 */
#define VI_ERROR_LINE_IN_USE        (_VI_ERROR+0x3FFF0042L) /* BFFF0042, -
1073807294 */
#define VI_ERROR_NSUP_MODE          (_VI_ERROR+0x3FFF0046L) /* BFFF0046, -
1073807290 */
#define VI_ERROR_SRQ_NOCCURRED      (_VI_ERROR+0x3FFF004AL) /* BFFF004A, -
1073807286 */
#define VI_ERROR_INV_SPACE          (_VI_ERROR+0x3FFF004EL) /* BFFF004E, -
1073807282 */
#define VI_ERROR_INV_OFFSET         (_VI_ERROR+0x3FFF0051L) /* BFFF0051, -
1073807279 */
#define VI_ERROR_INV_WIDTH          (_VI_ERROR+0x3FFF0052L) /* BFFF0052, -
1073807278 */
#define VI_ERROR_NSUP_OFFSET        (_VI_ERROR+0x3FFF0054L) /* BFFF0054, -
1073807276 */
#define VI_ERROR_NSUP_VAR_WIDTH     (_VI_ERROR+0x3FFF0055L) /* BFFF0055, -
1073807275 */
#define VI_ERROR_WINDOW_NMAPPED     (_VI_ERROR+0x3FFF0057L) /* BFFF0057, -
1073807273 */
#define VI_ERROR_RESP_PENDING       (_VI_ERROR+0x3FFF0059L) /* BFFF0059, -
1073807271 */
#define VI_ERROR_NLISTENERS         (_VI_ERROR+0x3FFF005FL) /* BFFF005F, -
1073807265 */
#define VI_ERROR_NCIC               (_VI_ERROR+0x3FFF0060L) /* BFFF0060, -
1073807264 */
#define VI_ERROR_NSYS_CNTLR         (_VI_ERROR+0x3FFF0061L) /* BFFF0061, -
1073807263 */
#define VI_ERROR_NSUP_OPER          (_VI_ERROR+0x3FFF0067L) /* BFFF0067, -
1073807257 */
#define VI_ERROR_INTR_PENDING       (_VI_ERROR+0x3FFF0068L) /* BFFF0068, -
1073807256 */
#define VI_ERROR_ASRL_PARITY        (_VI_ERROR+0x3FFF006AL) /* BFFF006A, -
1073807254 */
#define VI_ERROR_ASRL_FRAMING       (_VI_ERROR+0x3FFF006BL) /* BFFF006B, -
1073807253 */
#define VI_ERROR_ASRL_OVERRUN       (_VI_ERROR+0x3FFF006CL) /* BFFF006C, -
1073807252 */
#define VI_ERROR_TRIG_NMAPPED       (_VI_ERROR+0x3FFF006EL) /* BFFF006E, -
1073807250 */
#define VI_ERROR_NSUP_ALIGN_OFFSET  (_VI_ERROR+0x3FFF0070L) /* BFFF0070, -
1073807248 */
#define VI_ERROR_USER_BUF           (_VI_ERROR+0x3FFF0071L) /* BFFF0071, -
1073807247 */
#define VI_ERROR_RSRC_BUSY          (_VI_ERROR+0x3FFF0072L) /* BFFF0072, -
1073807246 */
#define VI_ERROR_NSUP_WIDTH         (_VI_ERROR+0x3FFF0076L) /* BFFF0076, -
1073807242 */
#define VI_ERROR_INV_PARAMETER      (_VI_ERROR+0x3FFF0078L) /* BFFF0078, -
1073807240 */
#define VI_ERROR_INV_PROT           (_VI_ERROR+0x3FFF0079L) /* BFFF0079, -
1073807239 */
#define VI_ERROR_INV_SIZE           (_VI_ERROR+0x3FFF007BL) /* BFFF007B, -
1073807237 */
#define VI_ERROR_WINDOW_MAPPED      (_VI_ERROR+0x3FFF0080L) /* BFFF0080, -
1073807232 */
#define VI_ERROR_NIMPL_OPER         (_VI_ERROR+0x3FFF0081L) /* BFFF0081, -
1073807231 */
#define VI_ERROR_INV_LENGTH         (_VI_ERROR+0x3FFF0083L) /* BFFF0083, -
1073807229 */
#define VI_ERROR_INV_MODE           (_VI_ERROR+0x3FFF0091L) /* BFFF0091, -
1073807215 */
#define VI_ERROR_SESN_NLOCKED       (_VI_ERROR+0x3FFF009CL) /* BFFF009C, -
1073807204 */
#define VI_ERROR_MEM_NSHARED        (_VI_ERROR+0x3FFF009DL) /* BFFF009D, -
1073807203 */
#define VI_ERROR_LIBRARY_NFOUND     (_VI_ERROR+0x3FFF009EL) /* BFFF009E, -
1073807202 */
```

```
#define VI_ERROR_NSUP_INTR          (_VI_ERROR+0x3FFF009FL) /* BFFF009F, -
1073807201 */
#define VI_ERROR_INV_LINE           (_VI_ERROR+0x3FFF00A0L) /* BFFF00A0, -
1073807200 */
#define VI_ERROR_FILE_ACCESS        (_VI_ERROR+0x3FFF00A1L) /* BFFF00A1, -
1073807199 */
#define VI_ERROR_FILE_IO            (_VI_ERROR+0x3FFF00A2L) /* BFFF00A2, -
1073807198 */
#define VI_ERROR_NSUP_LINE          (_VI_ERROR+0x3FFF00A3L) /* BFFF00A3, -
1073807197 */
#define VI_ERROR_NSUP_MECH          (_VI_ERROR+0x3FFF00A4L) /* BFFF00A4, -
1073807196 */
#define VI_ERROR_INTF_NUM_NCONFIG   (_VI_ERROR+0x3FFF00A5L) /* BFFF00A5, -
1073807195 */
#define VI_ERROR_CONN_LOST          (_VI_ERROR+0x3FFF00A6L) /* BFFF00A6, -
1073807194 */

/*- Event Types --------------------------------------------------------*/

#define VI_EVENT_IO_COMPLETION      (0x3FFF2009UL)
#define VI_EVENT_TRIG               (0xBFFF200AUL)
#define VI_EVENT_SERVICE_REQ        (0x3FFF200BUL)
#define VI_EVENT_CLEAR              (0x3FFF200DUL)
#define VI_EVENT_EXCEPTION          (0xBFFF200EUL)
#define VI_EVENT_GPIB_CIC           (0x3FFF2012UL)
#define VI_EVENT_GPIB_TALK          (0x3FFF2013UL)
#define VI_EVENT_GPIB_LISTEN        (0x3FFF2014UL)
#define VI_EVENT_VXI_VME_SYSFAIL    (0x3FFF201DUL)
#define VI_EVENT_VXI_VME_SYSRESET   (0x3FFF201EUL)
#define VI_EVENT_VXI_SIGP           (0x3FFF2020UL)
#define VI_EVENT_VXI_VME_INTR       (0xBFFF2021UL)
#define VI_EVENT_TCPIP_CONNECT      (0x3FFF2036UL)
#define VI_EVENT_USB_INTR           (0x3FFF2037UL)

#define VI_ALL_ENABLED_EVENTS       (0x3FFF7FFFUL)

/*- Other VISA Definitions ---------------------------------------------*/

#define VI_FIND_BUFLEN              (256)

#define VI_INTF_GPIB                (1)
#define VI_INTF_VXI                 (2)
#define VI_INTF_GPIB_VXI            (3)
#define VI_INTF_ASRL                (4)
#define VI_INTF_TCPIP               (6)
#define VI_INTF_USB                 (7)

#define VI_PROT_NORMAL              (1)
#define VI_PROT_FDC                 (2)
#define VI_PROT_HS488               (3)
#define VI_PROT_4882_STRS           (4)
#define VI_PROT_USBTMC_VENDOR       (5)

#define VI_FDC_NORMAL               (1)
#define VI_FDC_STREAM               (2)

#define VI_LOCAL_SPACE              (0)
#define VI_A16_SPACE                (1)
#define VI_A24_SPACE                (2)
#define VI_A32_SPACE                (3)
#define VI_OPAQUE_SPACE             (-1) /* 0xFFFF */

#define VI_UNKNOWN_LA               (-1)
#define VI_UNKNOWN_SLOT             (-1)
#define VI_UNKNOWN_LEVEL            (-1)

#define VI_QUEUE                    (1)
#define VI_HNDLR                    (2)
```

```
#define VI_SUSPEND_HNDLR            (4)
#define VI_ALL_MECH                 (-1) /* 0xFFFF */

#define VI_ANY_HNDLR                (0)

#define VI_TRIG_ALL                 (-2)
#define VI_TRIG_SW                  (-1)
#define VI_TRIG_TTL0                (0)
#define VI_TRIG_TTL1                (1)
#define VI_TRIG_TTL2                (2)
#define VI_TRIG_TTL3                (3)
#define VI_TRIG_TTL4                (4)
#define VI_TRIG_TTL5                (5)
#define VI_TRIG_TTL6                (6)
#define VI_TRIG_TTL7                (7)
#define VI_TRIG_ECL0                (8)
#define VI_TRIG_ECL1                (9)
#define VI_TRIG_PANEL_IN            (27)
#define VI_TRIG_PANEL_OUT           (28)

#define VI_TRIG_PROT_DEFAULT        (0)
#define VI_TRIG_PROT_ON             (1)
#define VI_TRIG_PROT_OFF            (2)
#define VI_TRIG_PROT_SYNC           (5)

#define VI_READ_BUF                 (1)
#define VI_WRITE_BUF                (2)
#define VI_READ_BUF_DISCARD         (4)
#define VI_WRITE_BUF_DISCARD        (8)
#define VI_IO_IN_BUF                (16)
#define VI_IO_OUT_BUF               (32)
#define VI_IO_IN_BUF_DISCARD        (64)
#define VI_IO_OUT_BUF_DISCARD       (128)

#define VI_FLUSH_ON_ACCESS          (1)
#define VI_FLUSH_WHEN_FULL          (2)
#define VI_FLUSH_DISABLE            (3)

#define VI_NMAPPED                  (1)
#define VI_USE_OPERS                (2)
#define VI_DEREF_ADDR               (3)

#define VI_TMO_IMMEDIATE            (0L)
#define VI_TMO_INFINITE             (-1L) /* 0xFFFFFFFFUL */

#define VI_NO_LOCK                  (0)
#define VI_EXCLUSIVE_LOCK           (1)
#define VI_SHARED_LOCK              (2)
#define VI_LOAD_CONFIG              (4)

#define VI_NO_SEC_ADDR              (-1) /* 0xFFFF */

#define VI_ASRL_PAR_NONE            (0)
#define VI_ASRL_PAR_ODD             (1)
#define VI_ASRL_PAR_EVEN            (2)
#define VI_ASRL_PAR_MARK            (3)
#define VI_ASRL_PAR_SPACE           (4)

#define VI_ASRL_STOP_ONE            (10)
#define VI_ASRL_STOP_ONE5           (15)
#define VI_ASRL_STOP_TWO            (20)

#define VI_ASRL_FLOW_NONE           (0)
#define VI_ASRL_FLOW_XON_XOFF       (1)
#define VI_ASRL_FLOW_RTS_CTS        (2)
#define VI_ASRL_FLOW_DTR_DSR        (4)

#define VI_ASRL_END_NONE            (0)
```

```
#define VI_ASRL_END_LAST_BIT        (1)
#define VI_ASRL_END_TERMCHAR        (2)
#define VI_ASRL_END_BREAK           (3)

#define VI_STATE_ASSERTED           (1)
#define VI_STATE_UNASSERTED         (0)
#define VI_STATE_UNKNOWN            (-1)

#define VI_BIG_ENDIAN               (0)
#define VI_LITTLE_ENDIAN            (1)

#define VI_DATA_PRIV                (0)
#define VI_DATA_NPRIV               (1)
#define VI_PROG_PRIV                (2)
#define VI_PROG_NPRIV               (3)
#define VI_BLCK_PRIV                (4)
#define VI_BLCK_NPRIV               (5)
#define VI_D64_PRIV                 (6)
#define VI_D64_NPRIV                (7)

#define VI_WIDTH_8                  (1)
#define VI_WIDTH_16                 (2)
#define VI_WIDTH_32                 (4)

#define VI_GPIB_REN_DEASSERT        (0)
#define VI_GPIB_REN_ASSERT          (1)
#define VI_GPIB_REN_DEASSERT_GTL    (2)
#define VI_GPIB_REN_ASSERT_ADDRESS  (3)
#define VI_GPIB_REN_ASSERT_LLO      (4)
#define VI_GPIB_REN_ASSERT_ADDRESS_LLO (5)
#define VI_GPIB_REN_ADDRESS_GTL     (6)

#define VI_GPIB_ATN_DEASSERT        (0)
#define VI_GPIB_ATN_ASSERT          (1)
#define VI_GPIB_ATN_DEASSERT_HANDSHAKE (2)
#define VI_GPIB_ATN_ASSERT_IMMEDIATE (3)

#define VI_GPIB_HS488_DISABLED      (0)
#define VI_GPIB_HS488_NIMPL         (-1)

#define VI_GPIB_UNADDRESSED         (0)
#define VI_GPIB_TALKER              (1)
#define VI_GPIB_LISTENER            (2)

#define VI_VXI_CMD16                (0x0200)
#define VI_VXI_CMD16_RESP16         (0x0202)
#define VI_VXI_RESP16               (0x0002)
#define VI_VXI_CMD32                (0x0400)
#define VI_VXI_CMD32_RESP16         (0x0402)
#define VI_VXI_CMD32_RESP32         (0x0404)
#define VI_VXI_RESP32               (0x0004)

#define VI_ASSERT_SIGNAL            (-1)
#define VI_ASSERT_USE_ASSIGNED      (0)
#define VI_ASSERT_IRQ1              (1)
#define VI_ASSERT_IRQ2              (2)
#define VI_ASSERT_IRQ3              (3)
#define VI_ASSERT_IRQ4              (4)
#define VI_ASSERT_IRQ5              (5)
#define VI_ASSERT_IRQ6              (6)
#define VI_ASSERT_IRQ7              (7)

#define VI_UTIL_ASSERT_SYSRESET     (1)
#define VI_UTIL_ASSERT_SYSFAIL      (2)
#define VI_UTIL_DEASSERT_SYSFAIL    (3)

#define VI_VXI_CLASS_MEMORY         (0)
#define VI_VXI_CLASS_EXTENDED       (1)
```

```
#define VI_VXI_CLASS_MESSAGE        (2)
#define VI_VXI_CLASS_REGISTER       (3)
#define VI_VXI_CLASS_OTHER          (4)

#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_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_PXI_STAR_TRIG_CONTROLLER (1413)

/*- Help Context ID Values ---------------------------------------------------*/

#define HlpCtxIConflictManager                  1450
#define HlpCtxIFormattedIO488                   1550
#define HlpCtxIResourceManager                  1650
#define HlpCtxIVendorResourceManager            1850
#define HlpCtxIVendorIO                         1950
#define HlpCtxIVisaSession                      2050
#define HlpCtxIMessage                          2150
#define HlpCtxIRegister                         2250
#define HlpCtxIGpib                             2350
#define HlpCtxISerial                           2450
#define HlpCtxITcpipInstr                       2550
#define HlpCtxIVxi                              2650
#define HlpCtxIUsb                              2750
#define HlpCtxIEvent                            2850
#define HlpCtxIEventManager                     2950
#define HlpCtxIBaseMessage                      3150
#define HlpCtxIAsyncMessage                     3250
#define HlpCtxISharedRegister                   3350
#define HlpCtxIGpibIntfc                        3450
#define HlpCtxIGpibIntfcMessage                 3550
#define HlpCtxITcpipSocket                      3650
#define HlpCtxIVxi3                             3750
#define HlpCtxIVxiMemacc                        3850
#define HlpCtxIVxiBackplane                     3950
#define HlpCtxIEventHandler                     4050
#define HlpCtxIEventIOCompletion                4250
#define HlpCtxIEventTrigger                     4350
#define HlpCtxIEventVxiSignal                   4450
#define HlpCtxIEventVxiVmeInterrupt             4550
#define HlpCtxIEventGpibCIC                     4650
#define HlpCtxIEventUsbInterrupt                4750
#define HlpCtxIResourceManager3                 4850
#define HlpCtxIPxi                              4950
#define HlpCtxIRegister64                       5050
#define HlpCtxISharedRegister64                 5150
#define HlpCtxIHislipInstr                      5250

#define HlpCtxEnumVisaStatusCode               10000
#define HlpCtxEnumConflictHandlerType          10200
#define HlpCtxEnumEventType                    10300
#define HlpCtxEnumHardwareInterfaceType        10400
#define HlpCtxEnumIOProtocol                   10500
#define HlpCtxEnumFDCMode                      10600
```

```
#define HlpCtxEnumAddressSpace                  10700
#define HlpCtxEnumEventMechanism                10800
#define HlpCtxEnumTriggerLine                   10900
#define HlpCtxEnumTriggerProtocol               11000
#define HlpCtxEnumBufferMask                    11100
#define HlpCtxEnumTimeout                       11200
#define HlpCtxEnumAccessMode                    11300
#define HlpCtxEnumSerialParity                  11400
#define HlpCtxEnumSerialStopBits                11500
#define HlpCtxEnumSerialFlowControl             11600
#define HlpCtxEnumSerialEndConst                11700
#define HlpCtxEnumLineState                     11800
#define HlpCtxEnumVXIMemoryAccessPrivilege      11900
#define HlpCtxEnumDataWidth                     12000
#define HlpCtxEnumRENControlConst               12100
#define HlpCtxEnumATNControlConst               12200
#define HlpCtxEnumGPIBAddressState              12300
#define HlpCtxEnumVXICommandQuery               12400
#define HlpCtxEnumAssertInterruptConst          12500
#define HlpCtxEnumAssertUtilityConst            12600
#define HlpCtxEnumVXIDevClass                   12700
#define HlpCtxEnumIEEEASCIIType                 12800
#define HlpCtxEnumIEEEBinaryType                12900
#define HlpCtxEnumPXIMemType                    13000

#define HlpCtxClsResourceManager                20000
#define HlpCtxClsFormattedIO488                 20100
#define HlpCtxClsVisaConflictTableManager       20200

#endif
```

8.3. Interface Hierarchy