KineticSystems Company, LLC

VISA Application Programming Interface

February 25, 2005

KineticSystems VISA Application Programming Interface

Table Of Contents

1 VISA Library

VISA (Virtual Instrument Software Architecture) provides an industry wide common I/O library API (Application Programming Interface) which is interoperable with a wide variety of instruments. Originally conceived for VXIbus applications, VISA has evolved into a specification that supports a wide variety of instrument platforms, including CompactPCI, TCP/IP and GPIB. KineticSystems VISA provides full software integration among instruments manufactured by KineticSystems and other vendors.

1.1 VISA Overview

This software package consists of two entities: a visa32 shared library that provides the management and I/O services described in the VISA specification, and a resource manager (resman) which queries the system and identifies the instruments present, and coordinates shared resources (such as device identifiers and memory spaces) among them. In addition, a user will require 1 or more KineticSystems VISA Plug-in packages. KineticSystems VISA supports a wide variety of instrument platforms including VXI, CompactPCI and TCP/IP; each platform has its own Plug-in package that provides platform-specific services for VISA. The VISA package is normally bundled with the appropriate Plug-in when an instrument is purchased from KineticSystems. Please see http://www.kscorp.com for the latest availability of the VISA package and associated Plug-ins.

1.2 Installation

The VISA package must be installed first, before any Plug-ins. If this is an upgrade of VISA, KineticSystems recommends that any previous Plug-ins and VISA layer be uninstalled before proceeding.

Insert the disk labled 'KineticSystems VISA' into a CD drive and run setup.exe; the setup script will guide you through the process of installation.

Consult the documentation that accompanied the hardware device or instrument for any special instructions regarding the installation of the VISA Plug-in.

2 Software Components

2.1 Resman – Resource Manager

After a system has first powered up, it is not operable until the VISA Resource manager (resman.exe) has run. Resman performs a number of instrument platform specific operations to probe the various components present. The results are recorded in the resource manager table, resman.tbl, by default located in the ResmanTables directory under the vxipnp installation directory. Because resman also writes out information to the various instruments themselves, it is very important that resman be run after any chassis or component of the system is power cycled.

Resman can be run either from the (Windows platform only) 'Start' button (Start→KineticSystems Visa→Resman→Resman), or from the command line. Resman offers a number of command line options which are only available if run from the command line:

```
   -v, --verbose
       produce detailed output

   -q, --quiet
       produce no output

   --nopause
       do not wait for keystroke to end

   -d, --directory=path
       specify the resman directory

   -l, --log
       output results to logfile (pesmanlog)

 --version
     print version

 -h, --help
     print this message
```

In addition to these options, each VISA Plug-in may insert its own options into those available to Resman; consult your Plug-in's documentation for information about additional Resman options.

The help option (`-h, --help`) can be used to find all options available with the current VISA/Plug-in configuration.

2.2 VISA Library Routines

The first time that any of the VISA calls are made, an initialization must be done. The VISA standard provides no interface for opening the VISA library or for initializing the VISA on a particular platform; this happens automatically when the first call is made to any VISA function.

The remainder of this chapter describes each VISA Library routine. These routines are all present in the visa.h file.

Users programming with VISA must use the visa32.lib file to link to the visa32.dll. Users must #include visa.h.

2.2.1 viClose

Syntax

```
ViStatus viClose(ViSession vi)
```

Purpose

Close the specified session, event, or find list.

Description

This function closes a session to a device, event or a find list. All data structures that had been allocated for the specified `vi` are freed.

Parameters

Parameter Name	Direction	Description
`vi`	Input	Unique logical identifier to a session.

Return Values

`VI_SUCCESS`	Session, event, or find list closed successfully.
`VI_ERROR_INV_SESSION`	The given session or object reference is invalid

2.2.2 viDisableEvent

Syntax

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

Purpose

Disable notification of an event type by the specified mechanisms.

Description

This operation disables servicing of an event identified by the `eventType` parameter for the mechanisms specified in the `mechanism` parameter. Specifying `VI_ALL_ENABLED_EVENTS` for the `eventType` parameter allows a session to stop receiving all events. The session can stop receiving queued events by specifying `VI_QUEUE`. Applications can stop receiving callback events by specifying either `VI_HNDLR` or `VI_SUSPEND_HNDLR`. Specifying `VI_ALL_MECH` disables both the queuing and callback mechanisms.

Parameters

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

Return Values

`VI_SUCCESS`	Event disabled successfully.
`VI_SUCCESS_EVENT_DIS`	Specified event is already disabled for at least one of the specified mechanisms.
`VI_ERROR_NSUP_OPER`	The given `vi` does not support this operation.
`VI_ERROR_INV_SESSION`	The given session or object reference is invalid.
`VI_ERROR_INV_EVENT`	Specified event type is not supported by the resource.
`VI_ERROR_INV_MECH`	Invalid mechanism specified.

2.2.3 viDiscardEvents

Syntax

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

Purpose

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

Description

This operation discards all pending occurrences of the specified event types and mechanisms from the specified session. The information about all the event occurrences that have not yet been handled is discarded. This operation is useful to remove event occurrences that an application no longer needs.

Parameters

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

Return Values

`VI_SUCCESS`	Event queue flushed successfully.
`VI_SUCCESS_QUEUE_EMPTY`	Operation completed successfully, but queue was empty.
`VI_ERROR_NSUP_OPER`	The given `vi` does not support this operation.
`VI_ERROR_INV_SESSION`	The given session or object reference is invalid.
`VI_ERROR_INV_EVENT`	Specified event type is not supported by the resource.
`VI_ERROR_INV_MECH`	Invalid mechanism specified.

2.2.4 viEnableEvent

Syntax

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

Purpose

Enable notification of a specified event.

Description

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

Parameters

Parameter Name	Direction	Description
`vi`	Input	Unique logical identifier to a session.
`eventType`	Input	Logical event identifier.
`mechanism`	Input	Specifies event handling mechanisms to be enabled. The queuing mechanism is enabled by specifying `VI_QUEUE`, and the callback mechanism is enabled by specifying `VI_HNDLR` or `VI_SUSPEND_HNDLR`. It is possible to enable both mechanisms simultaneously by specifying "bit-wise OR" of `VI_QUEUE` and one of the two mode values for the callback mechanism.
`Context`	Input	VI_NULL

Return Values

`VI_SUCCESS`	Event enabled successfully.
`VI_SUCCESS_EVENT_EN`	Specified event is already enabled for at least one of the specified mechanisms.
`VI_ERROR_NSUP_OPER`	The given `vi` does not support this operation.
`VI_ERROR_INV_SESSION`	The given session or object reference is invalid.
`VI_ERROR_INV_EVENT`	Specified event type is not supported by the resource.

VI_ERROR_INV_MECH	Invalid mechanism specified.
VI_ERROR_HNDLR_NINSTALLED	A handler is not currently installed for the specified event. The session cannot be enabled for the VI_HNDLR mode of the callback mechanism.

2.2.5 viEventHandler

Syntax

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

Purpose

Event service handler procedure prototype.

Description

This user handle is called whenever a session receives an event and is enabled for handling events in the `VI_HNDLR` mode. The user passes a reference to a function of their own matching this prototype to `viInstallHandler()`. The handler services the event and returns `VI_SUCCESS` on completion. Because each event type defines its own context in terms of attributes, refer to the appropriate event definition in the VISA specification to determine which attributes can be retrieved using the `context` parameter.

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

Normally, an application should return `VI_SUCCESS` from all callback handlers. Future versions of VISA may take actions based on other return values, so a user should return `VI_SUCCESS` from handlers.

Parameters

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

Return Values

`VI_SUCCESS`	Event handled successfully.

2.2.6 viFindNext

Syntax

```
ViStatus viFindNext(viFindList findList, ViRsrc instrDesc)
```

Purpose

Return the next resource found during a previous call to `viFindRsrc()`.

Description

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

Parameters

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

Return Values

`VI_SUCCESS`	Resource(s) found.
`VI_ERROR_INV_SESSION`	The given findList reference is invalid
`VI_ERROR_NSUP_OPER`	The given findList does not support this operation.
`VI_ERROR_RSRC_NFOUND`	There are no more matches.

2.2.7 viFindRsrc

Syntax

```
ViStatus viFindRsrc(ViSession vi,
                    ViString expr,
                    ViPFindList findList,
                    ViPUInt32 retCnt,
                    ViRsrc desc)
```

Purpose

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

Description

This operation matches the value specified in the `expr` parameter with the resources available for a particular interface. On successful completion, it returns the first resource found in the list and returns a count to indicate if there were more resources found for the designated interface. This function also returns a handle to a find list. This handle points to the list of resources and it must be used as an input to `viFindNext()`. When this handle is no longer needed, it should be passed to `viClose()`.

Both the findList and retCnt parameters are optional; either or both can be specified to VI_NULL, indicating findList handle and/or retCnt values should not be returned.

Parameters

Parameter Name	Direction	Description
`vi`	Input	Resource Manager session (should always be the Default Resource Manager for VISA returned from `viOpenDefaultRM()`).
`expr`	Input	This is a regular expression followed by an optional logical expression. The grammar for this expression is given below.
`findList`	Output	Returns a handle identifying this search session. This handle will be used as an input in `viFindNext()`.
`retCnt`	Output	Number of matches.
`desc`	Output	Returns a string identifying the location of a device. Strings can then be passed to `viOpen()` to establish a session to the given device.

Return Values

`VI_SUCCESS`	Resource(s) found.
`VI_ERROR_INV_SESSION`	The given session or object reference is invalid
`VI_ERROR_INV_EXPR`	Invalid expression specified for search.
`VI_ERROR_RSRC_NFOUND`	Specified expression does not match any devices.

2.2.7.1 Search Expression

The search criteria specified in the `expr` parameter has two parts: a regular expression over a resource string (which is explained below), and an optional logical expression over attribute values. The regular expression is matched against the resource strings of resources known to the VISA Resource Manager. If the resource string matches the regular expression, the attribute values of the resource are then matched against the expression over attribute values. If the match is successful, the resource has met the search criteria and gets added to the list of resources found.

2.2.7.1.1 Regular Expression

The regular expression over a resource string consists of ordinary characters as well as special characters. A regular expression is used for specifying patterns to match in a given string. Given a string and a regular expression, one can determine if the string matches the regular expression. A regular expression can also be used as a search criterion. Given a regular expression and a list of strings, one can match the regular expression against each string and return a list of strings that match the regular expression.

Character	Description	Symbol
NL / LF	New Line / Line Feed	"\n"
HT	Horizontal Tab	"\t"
CR	Carrage Return	"\r"
FF	Form Feed	"\f"
SP	Blank Space	" "

Special Characters

Literal	Definition
White_space	NL, LF, HT, CR, FF, SP
digit	"0","1".."9"
letter	"a","b".."z", "A","B".."Z"
Hex_digtit	"0","1".."9", "a","b".."f", "A","B".."F"
underscore	"_"

Literals

Special Characters and Operators	Meaning
?	Matches any one character.
\	Makes the character that follows it an ordinary character instead of special character. For example, when a question mark follows a backslash (i.e. '\?'), it matches the '?' character instead of any one character.
[list]	Matches any one character from the enclosed *list*. A hyphen can be used to match a range of characters.
[^ list]	Matches any character not in the enclosed *list*. A hyphen can be used to match a range of characters.
*	Matches 0 or more occurrences of the preceding character or expression.
+	Matches 1 or more occurrences of the preceding character or expression.

exp | exp	Matches either the preceding or following expression. The or operator | matches the entire expression that precedes or follows it and not just the character that precedes or follows it. For example, `VXI|GPIB` means `(VXI)|(GPIB)`, not `VXI(I|G)PIB`.
(exp)	Grouping characters or expressions.

Regular Expression Characters and Operators

2.2.7.1.2 Optional Attribute Expression

The optional attribute expression allows construction of flexible and powerful expressions with the use of logical ANDs, ORs and NOTs. Equal (==) and unequal (!=) comparators can be used compare attributes of any type, and in addition, other inequality comparators (>, <, >=, <=) can be used to compare attributes of numeric type. Only global attributes can be used in the attribute expression.

Special Character	Meaning
&&	Logical AND
||	Logical OR
!	Logical negation (NOT)
()	Parenthesis

Special Characters

```
expr :=
             regularExpr ['{' attrExpr '}']
attrExpr :=
             attrTerm |
             attrExpr '||' attrTerm
attrTerm :=
             attrFactor |
             attrTerm '&&' attrFactor
attrFactor :=
             '(' attrExpr ')' |
             '!' attrFactor |
             relationExpr

relationExpr :=
             attributeId compareOp numValue |
             attributeId equalityOp stringValue
compareOp :=
             '==' | '!=' | '>' | '<' | '>=' | '<='
equalityOp :=
             '==' | '!='
attributeId :=
                character (character|digit|underscore)*
numValue :=
             digit+ |
             '-' digit+ |
             '0x' hex_digit+ |
             '0X' hex_digit+
stringValue :=
             '"' character* '"'
```

2.2.7.2 Search expr Examples

expr	Sample Matches
VXI5::?*::INSTR	Matches all VXI instruments on board 5
VXI::2[0-9]*::INSTR	Matches all VXI instruments with logical address starting with '2' (2,21,256,2920, etc.)
PXI::2-[0-9]+::INSTR	Matches all PXI instruments on bus 2
?*INSTR	Matches all instruments
?*	Matches all resources
VXI::*?::INSTR {VI_ATTR_SLOT==5}	Matches all VXI instruments (in all chassis) located in slot 5
PXI::*?::INSTR {VI_ATTR_MANF_ID == 0x11f4 && VI_MODEL_CODE == 0x635}	Matches all PXI KineticSystems 635 instruments

2.2.8 viGetAttribute

Syntax

```
ViStatus viGetAttribute(ViSession vi,
                        ViAttr attribute,
                        ViPAttrState attrState)

ViStatus viGetAttribute(ViEvent vi,
                        ViAttr attribute,
                        ViPAttrState attrState)

ViStatus viGetAttribute(ViFindList vi,
                        ViAttr attribute,
                        ViPAttrState attrState)
```

Purpose

Retrieve the state of an attribute.

Description

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

Parameters

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

Return Values

`VI_SUCCESS`	Attribute retrieved successfully.
`VI_ERROR_INV_SESSION`	The given session or object reference is invalid
`VI_ERROR_NSUP_ATTR`	The specified attribute is not defined by the referenced session, event, or find list.

2.2.9 viIn8/viIn16/viIn32

Syntax

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

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

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

Purpose

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

Description

This operation, by using the specified address space, reads in 8, 16, or 32 bits of data from the specified offset. This operation does not require `viMapAddress()` to be called prior to its invocation.

Parameters

Parameter Name	Direction	Description
`vi`	Input	Unique logical identifier to a session.
`space`	Input	Specifies the address space.
`offset`	Input	Offset (in bytes) of the device to read from.
`val8, val16, val32`	Output	Data read from bus (8 bits for `viIn8()`,16 bits for `viIn16()`, and 32 bits for `viIn32()`).

Space	Description
`VI_A16_SPACE`	A16 address space of VXI/MXI bus
`VI_A24_SPACE`	A24 address space of VXI/MXI bus
`VI_A32_SPACE`	A32 address space of VXI/MXI bus
`VI_PXI_CFG_SPACE`	PXI Configuration space
`VI_PXI_BAR0_SPACE`	PXI BAR0 space
`VI_PXI_BAR1_SPACE`	PXI BAR1 space
`VI_PXI_BAR2_SPACE`	PXI BAR2 space
`VI_PXI_BAR3_SPACE`	PXI BAR3 space
`VI_PXI_BAR4_SPACE`	PXI BAR4 space
`VI_PXI_BAR5_SPACE`	PXI BAR5 space

Return Values

`VI_SUCCESS`	Operation completed successfully.
`VI_ERROR_INV_SESSION`	The given session or object reference is invalid
`VI_ERROR_NSUP_OPER`	The given `vi` does not support this operation.
`VI_ERROR_BERR`	Bus error occurred during transfer.
`VI_ERROR_INV_SPACE`	Invalid address space specified.
`VI_ERROR_INV_OFFSET`	Invalid offset specified.
`VI_ERROR_IO`	Device unavailable
`VI_ERROR_NSUP_OFFSET`	Specified offset is not accessible from this hardware.
`VI_ERROR_NSUP_WIDTH`	Specified width is not supported by this hardware.
`VI_ERROR_NSUP_ALIGN_OFFSET`	The specified offset is not properly aligned for the access width of the operation.

2.2.10 viInstallHandler

Syntax

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

Purpose

Install handlers for event callbacks.

Description

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

Parameters

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

Return Values

`VI_SUCCESS`	Event handler installed successfully.
`VI_ERROR_INV_SESSION`	The given session or object reference is invalid
`VI_ERROR_INV_EVENT`	Specified event type is not supported by the resource.
`VI_ERROR_INV_HNDLR_REF`	The given handler reference is invalid.
`VI_ERROR_HNDLR_REF`	The handler was not installed. This may be returned if an application attempts to install multiple handlers for the same event on the same session.

2.2.11 viMapAddress

Syntax

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

Purpose

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

Description

This operation maps in a specified memory space. The memory space that is mapped is dependent on the type of interface specified by the `vi` parameter and the `mapSpace` parameter. The `address` parameter returns the address in your process space where memory is mapped.

Parameters

Parameter Name	Direction	Description
`vi`	Input	Unique logical identifier to a session.
`mapSpace`	Input	Specifies the address space to map.
`mapBase`	Input	Offset (in bytes) of the memory to be mapped.
`mapSize`	Input	Amount of memory to map (in bytes).
`access`	Input	`VI_FALSE`
`suggested`	Input	`VI_NULL`
`address`	Output	Address in your process space where the memory was mapped.

Return Values

`VI_SUCCESS`	Map successful.
`VI_ERROR_INV_SESSION`	The given session or object reference is invalid
`VI_ERROR_NSUP_OPER`	The given `vi` does not support this operation.
`VI_ERROR_WINDOW_MAPPED`	The specified session already contains a mapped window.
`VI_ERROR_INV_SETUP`	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).
`VI_ERROR_INV_SPACE`	Invalid address space specified.
`VI_ERROR_INV_SIZE`	Invalid size of window specified.
`VI_ERROR_INV_OFFSET`	Invalid offset specified.
`VI_ERROR_INV_ACC_MODE`	Invalid access mode.
`VI_ERROR_TMO`	`viMapAddress()` could not acquire resource or perform mapping before the timer expired.

2.2.12 viMoveIn8/viMoveIn16/viMoveIn32

Syntax

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

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

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

Purpose

Move a block of data from the specified address space and offset to local memory in increments of 8, 16, or 32 bits.

Description

This operation, by using the specified address space, reads in 8, 16, or 32 bits of data from the specified offset. This operation does not require `viMapAddress()` to be called prior to its invocation.

Parameters

Parameter Name	Direction	Description
`vi`	Input	Unique logical identifier to a session.
`space`	Input	Specifies the address space.
`offset`	Input	Offset (in bytes) of the starting address to read
`length`	Input	Number of elements to transfer, where the data width of the elements to transfer is identical to data width (8, 16, or 32 bits).
`buf8, buf16, buf32`	Output	Data read from bus

Space	Description
`VI_A16_SPACE`	A16 address space of VXI/MXI bus
`VI_A24_SPACE`	A24 address space of VXI/MXI bus
`VI_A32_SPACE`	A32 address space of VXI/MXI bus
`VI_PXI_CFG_SPACE`	PXI Configuration space
`VI_PXI_BAR0_SPACE`	PXI BAR0 space

`VI_PXI_BAR1_SPACE`	PXI BAR1 space
`VI_PXI_BAR2_SPACE`	PXI BAR2 space
`VI_PXI_BAR3_SPACE`	PXI BAR3 space
`VI_PXI_BAR4_SPACE`	PXI BAR4 space
`VI_PXI_BAR5_SPACE`	PXI BAR5 space

Return Values

`VI_SUCCESS`	Operation completed successfully.
`VI_ERROR_INV_SESSION`	The given session or object reference is invalid
`VI_ERROR_NSUP_OPER`	The given `vi` does not support this operation.
`VI_ERROR_TMO`	Timeout expired before operation completed.
`VI_ERROR_BERR`	Bus error occurred during transfer.
`VI_ERROR_INV_SPACE`	Invalid address space specified.
`VI_ERROR_INV_OFFSET`	Invalid offset specified.
`VI_ERROR_NSUP_OFFSET`	Specified offset is not accessible from this hardware.
`VI_ERROR_NSUP_WIDTH`	Specified width is not supported by this hardware.
`VI_ERROR_INV_LENGTH`	Invalid length specified.
`VI_ERROR_NSUP_ALIGN_OFFSET`	The specified offset is not properly aligned for the access width of the operation.

2.2.13 viMoveOut8/viMoveOut16/viMoveOut32

Syntax

```
ViStatus viMoveOut8(ViSession vi,
                    ViUInt16 space,
                    ViBusAddress offset,
                    ViBusSize length,
                    ViAUInt8 buf8)

ViStatus viMoveOut16(ViSession vi,
                     ViUInt16 space,
                     ViBusAddress offset,
                     ViBusSize length,
                     ViAUInt16 buf16)

ViStatus viMoveOut32(ViSession vi,
                     ViUInt16 space,
                     ViBusAddress offset,
                     ViBusSize length,
                     ViAUInt32 buf32)
```

Purpose

Move a block of data from local memory to the specified address space and offset in increments of 8, 16, or 32 bits.

Description

This operation, by using the specified address space, writes 8, 16, or 32 bits of data to the specified offset. This operation does not require `viMapAddress()` to be called prior to its invocation.

Parameters

Parameter Name	Direction	Description
`vi`	Input	Unique logical identifier to a session.
`space`	Input	Specifies the address space.
`offset`	Input	Offset (in bytes) of the starting address to read
`length`	Input	Number of elements to transfer, where the data width of the elements to transfer is identical to data width (8, 16, or 32 bits).
`buf8, buf16, buf32`	Input	Data to write to bus.

Space	Description
`VI_A16_SPACE`	A16 address space of VXI/MXI bus
`VI_A24_SPACE`	A24 address space of VXI/MXI bus
`VI_A32_SPACE`	A32 address space of VXI/MXI bus
`VI_PXI_CFG_SPACE`	PXI Configuration space
`VI_PXI_BAR0_SPACE`	PXI BAR0 space

`VI_PXI_BAR1_SPACE`	PXI BAR1 space
`VI_PXI_BAR2_SPACE`	PXI BAR2 space
`VI_PXI_BAR3_SPACE`	PXI BAR3 space
`VI_PXI_BAR4_SPACE`	PXI BAR4 space
`VI_PXI_BAR5_SPACE`	PXI BAR5 space

Return Values

`VI_SUCCESS`	Operation completed successfully.
`VI_ERROR_INV_SESSION`	The given session or object reference is invalid
`VI_ERROR_NSUP_OPER`	The given `vi` does not support this operation.
`VI_ERROR_TMO`	Timeout expired before operation completed.
`VI_ERROR_BERR`	Bus error occurred during transfer.
`VI_ERROR_INV_SPACE`	Invalid address space specified.
`VI_ERROR_INV_OFFSET`	Invalid offset specified.
`VI_ERROR_NSUP_OFFSET`	Specified offset is not accessible from this hardware.
`VI_ERROR_NSUP_WIDTH`	Specified width is not supported by this hardware.
`VI_ERROR_INV_LENGTH`	Invalid length specified.
`VI_ERROR_NSUP_ALIGN_OFFSET`	The specified offset is not properly aligned for the access width of the operation.

2.2.14 viOpen

Syntax

```
ViStatus viOpen(ViSession sesn,
              ViRsrc rsrcName,
              ViAccessMode accessMode,
              ViUInt32 timeout,
              ViPSession vi)
```

Purpose

Open a session to the specified device.

Description

This operation opens a session to the specified device. It returns a session identifier that can be used to call any other operations of that device.

The grammar for the Address String is shown in below. Optional string segments are shown in square brackets ([]).

Interface	Grammar
VXI	VXI[*board*][::*node*]::*VXI logical Address*[::INSTR]
PXI	PXI[*inteface*]::*bus-device*[.*function*][::INSTR]
PXI	PXI[*bus*]::*device*[::*function*][::INSTR]
SC	SC[::*node*]::*slot*[::INSTR]

Parameters

Parameter Name	Direction	Description
`sesn`	Input	Resource Manager session (should always be the Default Resource Manager for VISA returned from viOpenDefaultRM()).
`rsrcName`	Input	Unique symbolic name of a resource.
`accessMode`	Input	VI_NULL
`timeout`	Input	VI_NULL
`vi`	Output	Unique logical identifier reference to a session.

Return Values

`VI_SUCCESS`	Session opened successfully.
`VI_SUCCESS_DEV_NPRESENT`	Session opened successfully, but the device at the specified address is not responding.
`VI_ERROR_INV_SESSION`	The given session or object reference is invalid
`VI_ERROR_NSUP_OPER`	The given `sesn` does not support this operation. For VISA, this operation is supported only by the Default Resource Manager session.

`VI_ERROR_TMO`	Timeout expired before operation completed.
`VI_ERROR_INV_RSRC_NAME`	Invalid resource reference specified. Parsing error.
`VI_ERROR_RSRC_NFOUND`	Insufficient location information or resource not present in the system.
`VI_ERROR_ALLOC`	Insufficient system resources to open a session.

2.2.15 viOpenDefaultRM

Syntax

```
ViStatus viOpenDefaultRM(ViPSession sesn)
```

Purpose

Return a session to the Default Resource Manager resource.

Description

This function must be called before any VISA operations can be invoked. The first call to this function initializes the VISA system, including the Default Resource Manager resource, and also returns a session to that resource. Subsequent calls to this function return unique sessions to the same Default Resource Manager resource.

Parameters

Parameter Name	Direction	Description
`sesn`	Output	Unique logical identifier to a Default Resource Manager session.

Return Values

`VI_SUCCESS`	Session to the Default Resource Manager resource created successfully.
`VI_ERROR_ALLOC`	Insufficient system resources to create a session to the Default Resource Manager resource.
`VI_ERROR_SYSTEM_ERROR`	The VISA system failed to initialize.

2.2.16 viOut8/viOut16/viOut32

Syntax

```
ViStatus viOut8(ViSession vi,
              ViUInt16 space,
              ViBusAddress offset,
              ViUInt8 val8)

ViStatus viOut16(ViSession vi,
               ViUInt16 space,
               ViBusAddress offset,
               ViUInt16 val16)

ViStatus viOut32(ViSession vi,
               ViUInt16 space,
               ViBusAddress offset,
               ViUInt32 val32)
```

Purpose

Write an 8-bit, 16-bit, or 32-bit value to the specified memory space and offset.

Description

This operation, by using the specified address space, writes 8, 16, or 32 bits of data to the specified offset. This operation does not require `viMapAddress()` to be called prior to its invocation.

Parameters

Parameter Name	Direction	Description
`vi`	Input	Unique logical identifier to a session.
`space`	Input	Specifies the address space.
`offset`	Input	Offset (in bytes) of the device to read from.
`val8, val16, val32`	Input	Data to write to bus (8 bits for `viOut8()`, 16 bits for `viOut16()`, and 32 bits for `viOut32()`).

Space	Description
`VI_A16_SPACE`	A16 address space of VXI/MXI bus
`VI_A24_SPACE`	A24 address space of VXI/MXI bus
`VI_A32_SPACE`	A32 address space of VXI/MXI bus
`VI_PXI_CFG_SPACE`	PXI Configuration space
`VI_PXI_BAR0_SPACE`	PXI BAR0 space
`VI_PXI_BAR1_SPACE`	PXI BAR1 space
`VI_PXI_BAR2_SPACE`	PXI BAR2 space
`VI_PXI_BAR3_SPACE`	PXI BAR3 space
`VI_PXI_BAR4_SPACE`	PXI BAR4 space
`VI_PXI_BAR5_SPACE`	PXI BAR5 space

Return Values

VI_SUCCESS	Operation completed successfully.
VI_ERROR_INV_SESSION	The given session or object reference is invalid
VI_ERROR_NSUP_OPER	The given `vi` does not support this operation.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SPACE	Invalid address space specified.
VI_ERROR_INV_OFFSET	Invalid offset specified.
VI_ERROR_IO	Device unavailable
VI_ERROR_NSUP_OFFSET	Specified offset is not accessible from this hardware.
VI_ERROR_NSUP_WIDTH	Specified width is not supported by this hardware.
VI_ERROR_NSUP_ALIGN_OFFSET	The specified offset is not properly aligned for the access width of the operation.

2.2.17 viRead

Syntax

```
ViStatus viRead(ViSession vi,
              ViPBuf buf,
              ViUInt32 cnt,
              ViPUInt32 retCnt)
```

Purpose

Read data from device synchronously.

Description

The synchronous read operation synchronously transfers data. The data read is to be stored in the buffer represented by `buf`. This operation returns only when the transfer terminates. Only one synchronous read operation can occur at any one time.

Parameters

Parameter Name	Direction	Description
`vi`	Input	Unique logical identifier to a session.
`buf`	Output	Represents the location of a buffer to receive data from a device.
`cnt`	Input	Number of bytes to be read.
`retCnt`	Output	Represents the location of an integer that will be set to the number of bytes actually transferred.

Return Values

`VI_SUCCESS`	The operation completed successfully and the END indicator was received.
`VI_SUCCESS_TERM_CHAR`	The specified termination character was read.
`VI_SUCCESS_MAX_CNT`	The number of bytes read is equal to `count`.
`VI_ERROR_INV_SESSION`	The given session or object reference is invalid
`VI_ERROR_NSUP_OPER`	The given `vi` does not support this operation.
`VI_ERROR_TMO`	Timeout expired before operation completed.
`VI_ERROR_RAW_WR_PROT_VIOL`	Violation of raw write protocol occurred during transfer.
`VI_ERROR_RAW_RD_PROT_VIOL`	Violation of raw read protocol occurred during transfer.
`VI_ERROR_OUTP_PROT_VIOL`	Device reported an output protocol error during transfer.
`VI_ERROR_BERR`	Bus error occurred during transfer.
`VI_ERROR_NCIC`	The interface associated with the given `vi` is not currently the controller in charge.
`VI_ERROR_NLISTENERS`	No Listeners condition is detected (both NRFD and NDAC are deasserted).
`VI_ERROR_IO`	An unknown I/O error occurred during transfer.

2.2.18 viSetAttribute

Syntax

```
ViStatus viSetAttribute(ViSession vi,
                        ViAttr attrName,
                        ViAttrState attrValue)
```

Purpose

Set the state of an attribute.

Description

The `viSetAttribute()` operation is used to modify the state of an attribute for the specified session.

Parameters

Parameter Name	Direction	Description
`vi`	Input	Unique logical identifier to a session.
`attrName`	Input	Session for which the state is modified.
`attrValue`	Input	The state of the attribute to be set for the specified resource. The interpretation of the individual attribute value is defined by the resource.

Return Values

VI_SUCCESS	Attribute value set successfully.
VI_ERROR_INV_SESSION	The given session or object reference is invalid
VI_ERROR_NSUP_ATTR	The specified attribute is not defined by the referenced session.
VI_ERROR_NSUP_ATTR_STATE	The specified state of the attribute is not valid, or is not supported as defined by the session.
VI_ERROR_ATTR_READONLY	The specified attribute is read-only.

2.2.19 viStatusDesc

Syntax

```
ViStatus viStatusDesc(ViSession vi,
                      ViStatus status,
                      ViPString desc)
```

Purpose

Return a user-readable description of the status code passed to the operation.

Description

The `viStatusDesc()` operation is used to retrieve a user-readable string that describes the status code presented.

Parameters

Parameter Name	Direction	Description
`vi`	Input	Unique logical identifier to a session.
`status`	Input	Status code to interpret.
`desc`	Output	The user-readable string interpretation of the status code passed to the operation.

Return Values

`VI_SUCCESS`	Description successfully returned.
`VI_WARN_UNKNOWN_STATUS`	The status code passed to the operation could not be interpreted.

2.2.20 viUninstallHandler

Syntax

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

Purpose

Uninstall handlers for events.

Description

This operation allows client applications to uninstall handlers for events on sessions. Applications should also specify the value in the `userHandle` parameter that was passed while installing the handler. VISA identifies handlers uniquely using the handler reference and this value. All the handlers, for which the handler reference and the value matches, are uninstalled. If VI_ANY_HNDLR is passed in as the value of handler, all the handlers with the matching value in the `userHandle` parameter are uninstalled.

Parameters

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

Return Values

`VI_SUCCESS`	Event handler successfully uninstalled.
`VI_ERROR_INV_SESSION`	The given session or object reference is invalid
`VI_ERROR_NSUP_OPER`	The given `vi` does not support this operation.
`VI_ERROR_INV_EVENT`	Specified event type is not supported by the resource.
`VI_ERROR_INV_HNDLR_REF`	Either the specified handler reference or the user context value (or both) does not match any installed handler.
`VI_ERROR_HNDLR_NINSTALLED`	A handler is not currently installed for the specified event.

2.2.21 viUnmapAddress

Syntax

```
ViStatus viUnmapAddress(ViSession vi)
```

Purpose

Unmap memory space previously mapped by `viMapAddress()`.

Description

This operation unmaps the region previously mapped by the `viMapAddress()` operation.

Parameters

Parameter Name	Direction	Description
`vi`	Input	Unique logical identifier to a session.

Return Values

`VI_SUCCESS`	Operation completed successfully.
`VI_ERROR_INV_SESSION`	The given session or object reference is invalid
`VI_ERROR_NSUP_OPER`	The given `vi` does not support this operation.
`VI_ERROR_WINDOW_NMAPPED`	The specified session is not currently mapped.

2.2.22 viWaitOnEvent

Syntax

```
ViStatus viWaitOnEvent(ViSession vi,
                       ViEventType inEventType,
                       ViUInt32 timeout,
                       ViPEventType outEventType,
                       ViPEvent outContext)
```

Purpose

Wait for an occurrence of the specified event for a given session.

Description

The `viWaitOnEvent()` operation suspends execution of a thread of application and waits for an event `inEventType` for a time period not to exceed that specified by `timeout`. Refer to individual event descriptions for context definitions. If the specified `inEventType` is `VI_ALL_ENABLED_EVENTS`, the operation waits for any event that is enabled for the given session. If the specified timeout value is `VI_TMO_INFINITE`, the operation is suspended indefinitely. If the specified timeout value is `VI_TMO_IMMEDIATE`, the operation will return immediately with an event (or error if no event is available) without suspending the application thread.

The `outEventType` and `outContext` parameters to the `viWaitOnEvent()` operation are optional (can be set to VI_NULL). This can be used if the event type is known from the `inEventType` parameter, or if the `eventContext` is not needed to retrieve additional information.

If a valid `outContext` is returned, the user application is responsible for closing it by passing it into `viClose()` when it is no longer needed.

Parameters

Parameter Name	Direction	Description
`vi`	Input	Unique logical identifier to a session.
`inEventType`	Input	Logical identifier of the event(s) to wait for.
`timeout`	Input	Absolute time period in time units that the resource shall wait for a specified event to occur before returning the time elapsed error. The time unit is in milliseconds.
`outEventType`	Output	Logical identifier of the event actually received.
`outContext`	Output	A handle specifying the unique occurrence of an event.

Return Values

`VI_SUCCESS`	Wait terminated successfully on receipt of an event occurrence. The queue is empty.
`VI_SUCCESS_QUEUE_NEMPTY`	Wait terminated successfully on receipt of an event notification. There is still at least one more event

	occurrence of the type specified by `inEventType` available for this session.
`VI_ERROR_INV_SESSION`	The given session or object reference is invalid
`VI_ERROR_NSUP_OPER`	The given `vi` does not support this operation.
`VI_ERROR_INV_EVENT`	The specified session is not currently mapped.
`VI_ERROR_TMO`	Specified event did not occur within the specified time period.
`VI_ERROR_NENABLED`	The session must be enabled for events of the specified type in order to receive them.

2.2.23 viWrite

Syntax

```
ViStatus viWrite(ViSession vi,
               ViBuf buf,
               ViUInt32 cnt,
               ViPUInt32 retcnt)
```

Purpose

Write data to device synchronously.

Description

The write operation synchronously transfers data. The data to be written is in the buffer represented by `buf`. This operation returns only when the transfer terminates. Only one synchronous write operation can occur at any one time.

Parameters

Parameter Name	Direction	Description
`vi`	Input	Unique logical identifier to a session.
`buf`	Input	Represents the location of a data block to be sent to device.
`cnt`	Input	Number of bytes to be written.
`retcnt`	Output	Represents the location of an integer that will be set to the number of bytes actually transferred.

Return Values

`VI_SUCCESS`	Transfer completed.
`VI_ERROR_INV_SESSION`	The given session or object reference is invalid
`VI_ERROR_NSUP_OPER`	The given `vi` does not support this operation.
`VI_ERROR_TMO`	Timeout expired before operation completed.
`VI_ERROR_RAW_WR_PROT_VIOL`	Violation of raw write protocol occurred during transfer.
`VI_ERROR_RAW_RD_PROT_VIOL`	Violation of raw read protocol occurred during transfer.
`VI_ERROR_INP_PROT_VIOL`	Device reported an input protocol error during transfer.
`VI_ERROR_BERR`	Bus error occurred during transfer.
`VI_ERROR_NCIC`	The interface associated with the given `vi` is not currently the controller in charge.
`VI_ERROR_NLISTENERS`	No Listeners condition is detected (both NRFD and NDAC are deasserted).
`VI_ERROR_IO`	An unknown I/O error occurred during transfer.
`VI_ERROR_TMO`	Timeout expired before operation completed.

3 KineticSystems VISA Special Features

3.1 Debugging

KineticSystems VISA provides a *spy* mechanism for debugging. When activated, spy generates output for every VISA API function called by and application. The output consists of the function name, the arguments passed in, and the overall return value. Spy output is generated at the *end* of the API call; this is necessary in order to display the overall return value, which is of course not known until the operation completes.

When an argument to a function is a pointer, if the object pointed to is a string or a datatype of 32 bits or less, the pointer is dereferenced on output, so the value it points to is displayed, instead of the pointer itself, which is unlikely to be of much value. If the pointer points to a structure or an array, the pointer itself it simply displayed.

In Windows environments, spy output goes to the debugger, if the application is being run in a debugger. If the application is not running in a debugger, output goes to the *system debugger*, where it can be spooled using a tool like *dbgview*. Please see http://www.sysinternals.com/ntw2k/freeware/debugview.shtml for details on *dbgview*.

Even when spy output is completely turned off, trace level output is produced when hard or unexpected failures are encountered (a file that should exist could not be opened, a resource was not acquired, an OS call unexpectantly returned failure). Inexplicable failures can often be identified by watching the appropriate debug output when the failure occurs.

Spy functionality can be activated on either a global or per device scope. When activated globally, output is generated for all operations, all devices. Alternatively, spy can be activated on 1 or more individual devices; in this case, only operations that specifically involve the specified device(s) generate output.

'Device spy' is activated when the device is referenced by software. 'Global spy' is activated when any device is referenced, as well as when any defaultRM session is referenced. Therefore, an application which only references 1 device will have slightly different spy output depending on if activated by device or globally. For example, a VISA call to viOpendefaultRM() does not touch any device, but does touch a defaultRM session. Therefore, 'device spy' will produce no output for viOpenDefaultRM(), but global spy will. Device spy only produces output for operations which touch that device. Both device and global spy can be activated simultaneously, but this is probably unwise. Duplicate output will be generated (once when device is touched and another when the defaultRM session is touched), and it is hard to tell the two apart.

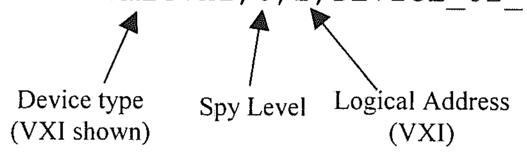
Activation of Global Spy

Spy functionality is activated on a global basis by modifying the registry key HKEY_LOCAL_MACHINE\SOFTWARE\KineticSystems, LLC\Pisa\Spy: '0' is off, '1' is on. The effect takes place the next time an application runs.

Activation of Per-Device Spy

Spy functionality is activated on a single device by hand-editing the resman table. Each line in the resman table represents 1 device; attributes of the device are separated by commas. Different device types have different formats (e.g., VXI vs. CPCI), but the first 2 fields are the same for all devices: the 1st field is a string representing the device type, and the 2nd field is a numeric value that represents the spy level. The value of '0' represents 'off and '1' represents 'on'. Activation of the spy level is done simply by opening the resman table in a text editor such as wordpad, changing the spy field on 1 or more devices, and saving the table.

```
vmicvxi,0,0,DEVICE_00_01,-1,0,0,f29,KineticSystems,...
vmicvxi,1,1,V208        ,-1,0,3,f29,KineticSystems,...
vmicvxi,0,2,DEVICE_02_01,-1,0,2,f29,KineticSystems,...
```



Spy activated for logical address 1

The next time the application runs, spy output is generated as specified. By default, resman writes out the spy level of all devices to '0' (off).

3.2 Services Directly offered via VISA Plug-ins

Some VISA Plug-in modules offer functionality and services beyond the scope of VISA. For example, a Plug-in which supports an instance of VXI may export functions that allow an application to directly interact with VXI hardware, bypassing VISA. To access this functionality, the user would simply link their application to the Plug-in library (or dynamically load the Plug-in library from the application). Please consult the documentation included with the Plug-in for information on any additional services offered.