**VISA Error Codes and Descriptions (contd)**

VI_ERROR_INV_OFFSET
*The offset specified is invalid*

VI_ERROR_INV_PROT
*The protocol specified is invalid.*

VI_ERROR_INV_RSRC_NAME
*The resources specified are invalid.*

VI_ERROR_INV_SESSION
*The session specified is invalid.*

VI_ERROR_INV_SETUP
*The setup specified is invalid. This can be due to attributes being set to an inconsistent state, or some other implementation-specific configuration file is corrupt or does not exist.*

VI_ERROR_INV_SIZE
*The specified window size is invalid.*

VI_ERROR_INV_SPACE
*The address space specified is invalid.*

VI_ERROR_IO
*Could not perform read/write function because of an I/O error, or an unknown I/O error occurred during transfer.*

VI_ERROR_LINE_IN_USE
*The specified trigger line is in use.*

VI_ERROR_MEM_NSHARED
*The device does not export any memory.*

VI_ERROR_NCIC
*The session is referring to something other than the controller in charge.*

VI_ERROR_NIMPL_OPER
*The given operation is not implemented.*

VI_ERROR_NLISTENERS
*No listeners are detected. (Both NRFD and NDAC are deasserted.)*

VI_ERROR_NSUP_ATTR
*The attribute specified is not supported by the specified resource.*

VI_ERROR_NSUP_ATTR_STATE
*The state specified for the attribute is not supported.*

VI_ERROR_NSUP_FMT
*The format specifier is not supported for the current argument type.*

VI_ERROR_NSUP_OFFSET
*The offset specified is not accessible.*

VI_ERROR_NSUP_OPER
*The operation specified is not supported in the given session.*

VI_ERROR_NSUP_WIDTH
*The specified width is not supported by this hardware.*

VI_ERROR_QUEUE_ERROR
*Unable to queue read or write operation.*

VI_ERROR_OUTP_PROT_VIOL
*Output protocol error occurred during transfer.*

VI_ERROR_RAW_RD_PROT_VIOL
*A violation of raw read protocol occurred during a transfer.*

VI_ERROR_RAW_WR_PROT_VIOL
*A violation of raw write protocol occurred during a transfer.*

VI_ERROR_RSRC_LOCKED
*The specified operation could not be performed because the resource identified by vi has been locked for this kind of access.*

**VISA Error Codes and Descriptions (contd)**

VI_ERROR_RSRC_NFOUND
*The expression specified does not match any device, or resource was not found*

VI_ERROR_SRQ_NOCCURED
*A service request has not been received for the session.*

VI_ERROR_SYSTEM_ERROR
*Unknown system error.*

VI_ERROR_TMO
*The operation failed to complete within the specified time period.*

VI_ERROR_WINDOW_MAPPED
*The specified session already contains a mapped window.*

VI_ERROR_WINDOW_NMAPPED
*The specified session is not currently mapped.*

# HP VISA

## Quick Reference Guide for C Programmers

HEWLETT® PACKARD



E2090-90030 Printed in U.S.A. E0596

## GPIB-VXI Interface

| Attribute Name | Data Type | Range |
|---|---|---|
| VI_ATTR_INTF_PARENT_NUM | ViUInt16 | 0 to FFFFh |

## VISA Event Attributes

| Attribute Name | Data Type | Range |
|---|---|---|
| VI_ATTR_BUFFER | ViBuf | N/A |
| VI_ATTR_EVENT_TYPE | ViEventType | VI_EVENT_SERVICE_REQ<br>VI_EVENT_VXI_SIGP<br>VI_EVENT_TRIG<br>VI_EVENT_IO_COMPLETION |
| VI_ATTR_JOB_ID | ViJobId | N/A |
| VI_ATTR_RECV_TRIG_ID | ViInt16 | VI_TRIG_TTL0 to VI_TRIG_TTL7<br>VI_TRIG_ECL0 to VI_TRIG_ECL1 |
| VI_ATTR_RET_COUNT | ViUInt32 | 0 to FFFFFFFFh |
| VI_ATTR_SIGP_STATUS_ID | ViUInt16 | 0 to FFFFh |
| VI_ATTR_STATUS | ViStatus | N/A |

## VISA Completion Codes and Descriptions

VI_SUCCESS
*Operation completed successfully.*

VI_SUCCESS_EVENT_DIS
*The specified event is already disabled.*

VI_SUCCESS_EVENT_EN
*The specified event is already enabled for at least one of the specified mechanisms.*

VI_SUCCESS_MAX_CNT
*The number of bytes specified were read.*

VI_SUCCESS_NESTED_EXCLUSIVE
*The specified access mode was successfully acquired, and this session has nested exclusive locks.*

VI_SUCCESS_NESTED SHARED
*The specified access mode was successfully acquired, and this session has nested shared locks.*

VI_SUCCESS_QUEUE_EMPTY
*The event queue was empty while trying to discard queued events.*

VI_SUCCESS_QUEUE_NEMPTY
*The event queue is not empty.*

VI_SUCCESS_SYNC
*The read or write operation performed synchronously.*

VI_SUCCESS_TERM_CHAR
*The specified termination character was read.*

VI_WARN_NSUP_ATTR_STATE
*The attribute state is not supported by this resource.*

VI_WARN_NSUP_BUF
*The specified buffer is not supported.*

VI_WARN_UNKNOWN_STATUS
*The status code passed to the function was unable to be interpreted.*

## VISA Error Codes and Descriptions

VI_ERROR_ALLOC
*Insufficient system resources to open a session or to allocate the buffer(s) or memory block of the specified size.*

VI_ERROR_ASRL_PARITY
*A parity error occurred during transfer.*

VI_ERROR_ASRL_FRAMING
*A framing error occurred during transfer.*

VI_ERROR_ASRL_OVERRUN
*An overrun error occurred during transfer. A character was not read from the hardware before the next character arrived.*

VI_ERROR_ATTR_READONLY
*The attribute specified is read-only.*

VI_ERROR_BERR
*A bus error occurred during transfer.*

VI_ERROR_CLOSING_FAILED
*Unable to deallocate the previously allocated data structures for this session.*

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

VI_ERROR_INP_PROT_VIOL
*Input protocol error occurred during transfer.*

VI_ERROR_INV_ACCESS_KEY
*The requestedKey value passed is not a valid access key to the specified resource.*

VI_ERROR_INV_ACC_MODE
*The access mode specified is invalid.*

VI_ERROR_INV_CONTEXT
*The event context specified is invalid.*

VI_ERROR_INV_DEGREE
*The specified degree is invalid.*

VI_ERROR_INV_EVENT
*The event type specified is invalid for the specified resource.*

VI_ERROR_INV_EXPR
*The expression specified is invalid.*

VI_ERROR_INV_FMT
*The format specifier is invalid for the current argument.*

VI_ERROR_INV_HNDLR_REF
*The specified handler reference and/or the user context value does not match the installed handler.*

VI_ERROR_INV_JOB_ID
*The specified job identifier is invalid.*

VI_ERROR_INV_LENGTH
*The length specified is invalid.*

VI_ERROR_INV_LOCK_TYPE
*The specified type of lock is not supported by this resource.*

VI_ERROR_INV_MASK
*The system cannot set the buffer for the given mask, or the specified mask does not specify a valid flush operation on the read/write resource.*

VI_ERROR_INV_MECH
*The mechanism specified for the event is invalid.*

VI_ERROR_INV_OBJECT
*The object reference is invalid.*

# HP Computer Museum
# www.hpmuseum.net

**For research and education purposes only.**

### Access/Resource Management

viOpenDefaultRM(ViPSession *sesn*);
*Open Default Resource Manager Session*

### Lifecycle

viOpen(ViSession *sesn*, ViRsrc *rsrcName*, ViAccessMode *accessMode*, ViUInt32 *timeout*, ViPSession *vi*);
*Open Session*

viClose(ViSession/ViEvent/ViFindList *vi*);
*Close Session*

### Asynchronous Operation Control

viTerminate (ViSession *vi*, ViUInt16 *degree*, ViJobId *jobId*);
*Terminate Asynchronous Operation.*

### Characteristic Control

viGetAttribute(ViSession/ViEvent/ViFindList *vi*, ViAttr *attribute*, ViPAttrState *attrState*);
*Get Attribute*

viSetAttribute(ViSession/ViEvent/ViFind List *vi*, ViAttr *attribute*, ViAttrState *attrState*);
*Set Attribute*

viStatusDesc(ViSession/ViEvent/ViFindList *vi*, ViStatus *status*, ViPString *desc*);
*Get Status Code Description*

### Access Control

viLock (ViSession *vi*, ViAccessMode *lockType*, ViUInt32 *timeout*, ViKeyId *requestKey*, ViPKeyId *accessKey*);
*Lock Resource*

viUnlock (ViSession *vi*);
*Unlock Resource.*

### Event Handling

viEnableEvent(ViSession *vi*, ViEventType *eventType*, ViUInt16 *mechanism*, ViEventFilter *context*);
*Enable Event*

viDisableEvent(ViSession *vi*, ViEventType *eventType*, ViUInt16 *mechanism*);
*Disable Event*

viDiscardEvents(ViSession *vi*, ViEventType *eventType*, ViUInt16 *mechanism*);
*Discard Events*

viWaitOnEvent(ViSession *vi*, ViEventType *inEventType*, ViUInt32 *timeout*, ViPEventType *outEventType*, ViPEvent *outContext*);
*Wait on Event*

viInstallHandler(ViSession *vi*, ViEventType *eventType*, ViHndlr *handler*, ViAddr *userHandle*);
*Install Handler*

viUninstallHandler(ViSession *vi*, ViEventType *eventType*, ViHndlr *handler*, ViAddr *userHandle*);
*Uninstall Handler*

viEventHandler(ViSession *vi*, ViEventType *eventType*, ViEvent *context*, ViAddr *userHandle*);
*Event Handler Prototype*

### Searching

viFindRsrc(ViSession *sesn*, ViString *expr*, ViPFindList *findList*, ViPUInt32 *retcnt*, ViPRsrc *instrDesc*);
*Find Device*

viFindNext(ViFindList *findList*, ViPRsrc *instrDesc*);
*Find Next Device*

### Basic I/O

viRead(ViSession *vi*, ViPBuf *buf*, ViUInt32 *count*, ViPUInt32 *retCount*);
*Read Data from Device*

viReadAsync (ViSession *vi*, ViPBuf *buf*, ViUInt32 *count*, ViPJobId *jobId*);
*Read Data Asynchronously from Device.*

viWrite(ViSession *vi*, ViBuf *buf*, ViUInt32 *count*, ViPUInt32 *retCount*);
*Write Data to Device*

viWriteAsync (ViSession *vi*, ViBuf *buf*, ViUInt32 *count*, ViPJobId *jobId*);
*Write Data Asynchronously to Device.*

viAssertTrigger(ViSession *vi*, ViUInt16 *protocol*);
*Read Status Byte*

viReadSTB(ViSession *vi*, ViPUInt16 *status*);
*Assert Software/Hardware Trigger*

viClear(ViSession *vi*);
*Clear a Device*

### Formatted I/O

viSetBuf(ViSession *vi*, *ViUInt16* mask, ViUInt32 *size*);
*Set Size of Buffer*

viFlush(ViSession *vi*, ViUInt16 *mask*);
*Flush Read and Write Buffers*

viPrintf(ViSession *vi*, ViString *writeFmt*, *arg1, arg2, ...*);
*Convert, Format, and Send Parameters*

viVPrintf(ViSession *vi*, ViString *writeFmt*, ViVAList *params*);
*Convert, Format, and Send Parameters*

viScanf(ViSession *vi*, ViString *readFmt*, *arg1, arg2, ...*);
*Read, Convert, Format, and Store Data*

viVScanf(ViSession *vi*, ViString *readFmt*, ViPVAList *params*);
*Read, Convert, Format, and Store Data*

viQueryf (ViSession *vi*, ViString *writeFmt*, ViString *readFmt*, *arg1, arg2, ...*);
*Write and Read Formatted Data.*

viVQueryf(ViSession *vi*, ViString *writeFmt*, ViString *readFmt*, ViVAList *params*);
*Write and Read Formatted Data.*

### Shared Memory

viMemAlloc (ViSession *vi*, ViBusSize *size*, ViPBusAddress *offset*);
*Allocate Memory*

viMemFree (ViSession *vi*, ViBusAddress *offset*);
*Free Memory Previously Allocated*

### Memory I/O

viIn8(ViSession *vi*, ViUInt16 *space*, ViBusAddress *offset*, ViPUInt8 *val8*);
*Read 8-bit Value from Memory Space*

viIn16(ViSession *vi*, ViUInt16 *space*, ViBusAddress *offset*, ViPUInt16 *val16*);
*Read 16-bit Value from Memory Space*

viIn32 (ViSession *vi*, ViUInt16 *space*, ViBusAddress *offset*, ViPUInt32 *val32*);
*Read 32-bit Value from Memory Space*

viOut8(ViSession *vi*, ViUInt16 *space*, ViBusAddress *offset*, ViUInt8 *val8*);
*Write 8-bit Value to Memory Space*

viOut16(ViSession *vi*, ViUInt16 *space*, ViBusAddress *offset*, ViUInt16 *val16*);
*Write 16-bit Value to Memory Space*

viOut32 (ViSession *vi*, ViUInt16 *space*, ViBusAddress *offset*, ViUInt32 *val32*);
*Write 32-bit Value to Memory Space*

viMoveIn8 (ViSession *vi*, ViUInt16 *space*, ViBusAddress *offset*, ViBusSize *length*, ViAUInt8 *buf8*);
*Move 8-bit Value from Device Memory to Local Memory*

viMoveIn16 (ViSession *vi*, ViUInt16 *space*, ViBusAddress *offset*, ViBusSize *length*, ViAUInt16 *buf16*);
*Move 16-bit Value from Device Memory to Local Memory*

viMoveIn32 (ViSession *vi*, ViUInt16 *space*, ViBusAddress *offset*, ViBusSize *length*, ViAUInt32 *buf32*);
*Move 32-bit Value from Device Memory to Local Memory*

viMoveOut8 (ViSession *vi*, ViUInt16 *space*, ViBusAddress *offset*, ViBusSize *length*, ViAUInt8 *buf8*);
*Move 8-bit Value from Local Memory to Device Memory*

viMoveOut16 (ViSession *vi*, ViUInt16 *space*, ViBusAddress *offset*, ViBusSize *length*, ViAUInt16 *buf16*);
*Move 16-bit Value from Local Memory to Device Memory*

viMoveOut32 (ViSession *vi*, ViUInt16 *space*, ViBusAddress *offset*, ViBusSize *length*, ViAUInt32 *buf32*);
*Move 32-bit Value from Local Memory to Device Memory*

viMapAddress(ViSession *vi*, ViUInt16 *mapSpace*, ViBusAddress *mapBase*, ViBusSize *mapSize*, ViBoolean *access*, ViAddr *suggested*, ViPAddr *address*);
*Map Memory Space*

viUnmapAddress(ViSession *vi*);
*Unmap Memory Space*

viPeek8(ViSession *vi*, ViAddr *addr*, ViPUInt8 *val8*);
*Read 8-bit Value from Address*

viPeek16(ViSession *vi*, ViAddr *addr*, ViPUInt16 *val16*);
*Read 16-bit Value from Address*

viPeek32 (ViSession *vi*, ViAddr *addr*, ViPUInt 32 *val32*);
*Read 32-bit Value from Address*

viPoke8(ViSession *vi*, ViAddr *addr*, ViUInt8 *val8*);
*Write 8-bit Value to Address*

viPoke16(ViSession *vi*, ViAddr *addr*, ViUInt16 *val16*);
*Write 16-bit Value to Address*

viPoke32 (ViSession *vi*, ViAddr *addr*, ViUInt32 *val32*);
*Write 32-bit Value to Address*

## VISA Resource Attributes

| Attribute Name Data | Type | Range |
|---|---|---|
| VI_ATTR_MAX_QUEUE_LENGTH | ViUInt32 | 1h to 32,767<br>(50 default) |
| VI_ATTR_RM_SESSION | ViSession | N/A |
| VI_ATTR_RSRC_IMPL_VERSION | ViVersion | 0h to FFFFFFFFh |
| VI_ATTR_RSRC_LOCK_STATE | ViAccessMode | VI_NO_LOCK (default)<br>VI_EXCLUSIVE_LOCK<br>VI_SHARED_LOCK |
| VI_ATTR_RSRC_MANF_ID | ViUInt16 | 0h to 3FFFh |
| VI_ATTR_RSRC_MANF_NAME | ViString | N/A |
| VI_ATTR_RSRC_NAME | ViRsrc | N/A |
| VI_ATTR_RSRC_SPEC_VERSION | ViVersion | 001000000h |
| VI_ATTR_USER_DATA | ViAddr | N/A |

## VISA Generic Instrument Attributes

| Attribute Name | Type | Range |
|---|---|---|
| VI_ATTR_INTF_NUM | ViUInt16 | 0 to FFFFh<br>(0 default) |
| VI_ATTR_INTF_TYPE | ViUInt16 | VI_INTF_VXI<br>VI_INTF_GPIB<br>VI_INTF_GPIB_VXI<br>VI_INTF_ASRL |
| VI_ATTR_IO_PROT | ViUInt16 | VI_NORMAL (default)<br>VI_FDC<br>VI_HS488 |
| VI_ATTR_RD_BUF_<br>OPER_MODE | ViUInt16 | VI_FLUSH_ON_ACCESS<br>VI_FLUSH_DISABLE (default) |
| VI_ATTR_SEND_END_EN | ViBoolean | VI_TRUE (default)<br>VI_FALSE |
| VI_ATTR_SUPPRESS_<br>END_EN | ViBoolean | VI_TRUE<br>VI_FALSE (default) |
| VI_ATTR_TERMCHAR | ViUInt8 | 0 to FFh (0Ah default) |
| VI_ATTR_TERMCHAR_EN | ViBoolean | VI_TRUE<br>VI_FALSE (default) |
| VI_ATTR_TMO_VALUE | ViUInt32 | VI_TMO_IMMEDIATE<br>I to FFFFFFFEh<br>VI_TMO_INFINITE<br>(2000 default) |
| VI_ATTR_TRIG_ID | ViInt16 | VI_TRIG_SW (default)<br>VI_TRIG_TTL0 to VI_TRIG_TTL7<br>VI_TRIG_ECL0 to VI_TRIG_ECLI |
| VI_ATTR_WR_BUF_<br>OPER_MODE | ViUInt16 | VI_FLUSH_ON_ACCESS<br>VI_FLUSH_WHEN_FULL (default) |

## GPIB and GPIB-VXI Interfaces

| Attribute Name | Data Type | Range |
|---|---|---|
| VI_ATTR_GPIB_PRIMARY_ADDR | ViUInt16 | 0 to 30 |
| VI_ATTR_GPIB_SECONDARY_ADDR | ViUInt16 | 0 to 31<br>VI_NO_SEC_ADDR |

## VXI and GPIB-VXI Interfaces

| Attribute Name | Data Type | Range |
|---|---|---|
| VI_ATTR_CMDR_LA | ViInt16 | 0 to 255 |
| VI_ATTR_DEST_INCREMENT | ViInt32 | 0 to 1 (1 is default) |
| VI_ATTR_FDC_CHNL | ViUInt16 | 0 to 7 |
| VI_ATTR_FDC_GEN_SIGNAL_EN | ViBoolean | VI_TRUE<br>VI_FALSE (default) |
| VI_ATTR_FDC_MODE | ViUInt16 | VI_FDC_NORMAL (default)<br>VI_FDC_STREAM |
| VI_ATTR_FDC_USE_PAIR | ViBoolean | VI_TRUE<br>VI_FALSE (default) |
| VI_ATTR_IMMEDIATE_SERV | ViBoolean | VI_TRUE<br>VI_FALSE |
| VI_ATTR_MAINFRAME_LA | ViInt16 | 0 to 255<br>VI_UNKNOWN_LA |
| VI_ATTR_MANF_ID | ViUInt16 | 0 to FFFh |
| VI_ATTR_MEM_BASE | ViBussAddress | 0 to VI_VXI_MAX_ADDR |
| VI_ATTR_MEM_SIZE | ViBusSize | 0 to VI_VXI_MAX_SIZE |
| VI_ATTR_MEM_SPACE | ViUInt16 | VI_A16_SPACE (default)<br>VI_A24_SPACE<br>VI_A32_SPACE |
| VI_ATTR_MODEL_CODE | ViUInt16 | 0 to FFFFh |
| VI_ATTR_SLOT | ViInt16 | 0 to 12<br>VI_UNKNOWN_SLOT |
| VI_ATTR_SRC_INCREMENT | ViInt32 | 0 to 1 (1 is default) |
| VI_ATTR_VXI_LA | ViInt16 | 0 to 255 |
| VI_ATTR_WIN_ACCESS | ViUInt16 | VI_NMAPPED<br>VI_USE_OPERS<br>VI_DEREF_ADDR |
| VI_ATTR_WIN_BASE_ADDR | ViBusAddress | N/A |
| VI_ATTR_WIN_SIZE | ViBusSize | N/A |

## ASRL Interface

| Attribute Name | Data Type | Range |
|---|---|---|
| VI_ATTR_ASRL_AVAIL_NUM | ViUInt32 | 0 to FFFFFFFFh |
| VI_ATTR_ASRL_BAUD | ViUInt32 | 0 to FFFFFFFFh (9600 default) |
| VI_ATTR_ASRL_DATA_BITS | ViUInt16 | 5 to 8 (8default) |
| VI_ATTR_ASRL_END_IN | ViUInt16 | VI_ASRL_END_NONE<br>VI_ASRL_END_LAST_BIT<br>VI_ASRL_END_TERMCHAR (def) |
| VI_ATTR_ASRL_END_OUT | ViUInt16 | VI_ASRL_END_NONE (def)<br>VI_ASRL_END_LAST_BIT<br>VI_ASRL_END_BREAK |
| VI_ATTR_ASRL_FLOW_CNTRL | ViUInt16 | VI_ASRL_FLOW_NONE (def)<br>VI_ASRL_FLOW_XON_XOFF<br>VI_ASRL_FLOW_RTS_CTS |
| VI_ATTR_ASRL_PARITY | ViUInt16 | VI_ASRL_PAR_NONE (def)<br>VI_ASRL_PAR_ODD<br>VI_ASRL_PAR_EVEN<br>VI_ASRL_PAR_MARK<br>VI_ASRL_PAR_SPACE |
| VI_ATTR_ASRL_STOP_BITS | ViUInt16 | VI_ASRL_STOP_ONE (def)<br>VI_ASRL_STOP_TWO |