

---

# NI-VISA API Reference

---

2024-08-13



# Contents

NI-VISA API Reference .....	10
Attributes .....	11
VI_ATTR_4882_COMPLIANT .....	11
VI_ATTR_ASRL_ALLOW_TRANSMIT .....	11
VI_ATTR_ASRL_AVAIL_NUM .....	12
VI_ATTR_ASRL_BAUD .....	13
VI_ATTR_ASRL_BREAK_LEN .....	14
VI_ATTR_ASRL_BREAK_STATE .....	15
VI_ATTR_ASRL_CONNECTED .....	16
VI_ATTR_ASRL_CTS_STATE .....	16
VI_ATTR_ASRL_DATA_BITS .....	17
VI_ATTR_ASRL_DCD_STATE .....	18
VI_ATTR_ASRL_DISCARD_NULL .....	19
VI_ATTR_ASRL_DSR_STATE .....	20
VI_ATTR_ASRL_DTR_STATE .....	21
VI_ATTR_ASRL_END_IN .....	22
VI_ATTR_ASRL_END_OUT .....	23
VI_ATTR_ASRL_FLOW_CNTRL .....	25
VI_ATTR_ASRL_PARITY .....	27
VI_ATTR_ASRL_REPLACE_CHAR .....	28
VI_ATTR_ASRL_RI_STATE .....	28
VI_ATTR_ASRL_RTS_STATE .....	29
VI_ATTR_ASRL_STOP_BITS .....	31
VI_ATTR_ASRL_WIRE_MODE .....	31
VI_ATTR_ASRL_XOFF_CHAR .....	33
VI_ATTR_ASRL_XON_CHAR .....	34
VI_ATTR_BUFFER .....	35
VI_ATTR_CMDR_LA .....	36
VI_ATTR_DEST_ACCESS_PRIV .....	36
VI_ATTR_DEST_BYTE_ORDER .....	37
VI_ATTR_DEST_INCREMENT .....	38
VI_ATTR_DEV_STATUS_BYTE .....	39

VI_ATTR_DMA_ALLOW_EN .....	40
VI_ATTR_EVENT_TYPE .....	41
VI_ATTR_FDC_CHNL .....	42
VI_ATTR_FDC_MODE .....	43
VI_ATTR_FDC_USE_PAIR .....	43
VI_ATTR_FILE_APPEND_EN .....	44
VI_ATTR_GPIB_ADDR_STATE .....	45
VI_ATTR_GPIB_ATN_STATE .....	46
VI_ATTR_GPIB_CIC_STATE .....	47
VI_ATTR_GPIB_HS488_CBL_LEN .....	48
VI_ATTR_GPIB_NDAC_STATE .....	48
VI_ATTR_GPIB_PRIMARY_ADDR .....	49
VI_ATTR_GPIB_READDR_EN .....	50
VI_ATTR_GPIB_RECV_CIC_STATE .....	51
VI_ATTR_GPIB_REN_STATE .....	52
VI_ATTR_GPIB_SECONDARY_ADDR .....	53
VI_ATTR_GPIB_SRQ_STATE .....	54
VI_ATTR_GPIB_SYS_CNTRL_STATE .....	55
VI_ATTR_GPIB_UNADDR_EN .....	55
VI_ATTR_IMMEDIATE_SERV .....	56
VI_ATTR_INTF_INST_NAME .....	57
VI_ATTR_INTF_NUM .....	58
VI_ATTR_INTF_TYPE .....	59
VI_ATTR_INTR_STATUS_ID .....	60
VI_ATTR_IO_PROT .....	61
VI_ATTR_JOB_ID .....	62
VI_ATTR_MAINFRAME_LA .....	63
VI_ATTR_MANF_ID .....	64
VI_ATTR_MANF_NAME .....	65
VI_ATTR_MAX_QUEUE_LENGTH .....	66
VI_ATTR_MEM_BASE/VI_ATTR_MEM_BASE_32/VI_ATTR_MEM_BASE_64 .....	67
VI_ATTR_MEM_SIZE/VI_ATTR_MEM_SIZE_32/VI_ATTR_MEM_SIZE_64 .....	68
VI_ATTR_MEM_SPACE .....	69
VI_ATTR_MODEL_CODE .....	70
VI_ATTR_MODEL_NAME .....	71
VI_ATTR_OPER_NAME .....	72

VI_ATTR_PXI_ACTUAL_LWIDTH.....	73
VI_ATTR_PXI_BUS_NUM.....	74
VI_ATTR_PXI_CHASSIS .....	74
VI_ATTR_PXI_DEST_TRIG_BUS.....	75
VI_ATTR_PXI_DEV_NUM .....	76
VI_ATTR_PXI_DSTAR_BUS .....	77
VI_ATTR_PXI_DSTAR_SET.....	78
VI_ATTR_PXI_FUNC_NUM .....	79
VI_ATTR_PXI_IS_EXPRESS .....	80
VI_ATTR_PXI_MAX_LWIDTH .....	80
VI_ATTR_PXI_MEM_BASE_BAR0/1/2/3/4/5 .....	81
VI_ATTR_PXI_MEM_SIZE_BAR0/1/2/3/4/5 .....	82
VI_ATTR_PXI_MEM_TYPE_BAR0/1/2/3/4/5.....	83
VI_ATTR_PXI_RECV_INTR_DATA .....	84
VI_ATTR_PXI_RECV_INTR_SEQ.....	84
VI_ATTR_PXI_SLOT_LBUS_LEFT.....	85
VI_ATTR_PXI_SLOT_LBUS_RIGHT .....	86
VI_ATTR_PXI_SLOT_LWIDTH .....	87
VI_ATTR_PXI_SLOTPATH.....	88
VI_ATTR_PXI_SRC_TRIG_BUS .....	89
VI_ATTR_PXI_STAR_TRIG_BUS .....	90
VI_ATTR_PXI_STAR_TRIG_LINE.....	91
VI_ATTR_PXI_TRIG_BUS.....	92
VI_ATTR_RD_BUF_OPER_MODE.....	93
VI_ATTR_RD_BUF_SIZE .....	94
VI_ATTR_RECV_INTR_LEVEL .....	95
VI_ATTR_RECV_TRIG_ID .....	96
VI_ATTR_RET_COUNT/VI_ATTR_RET_COUNT_32/VI_ATTR_RET_COUNT_64 .....	97
VI_ATTR_RM_SESSION .....	98
VI_ATTR_RSRC_CLASS .....	99
VI_ATTR_RSRC_IMPL_VERSION .....	99
VI_ATTR_RSRC_LOCK_STATE .....	100
VI_ATTR_RSRC_MANF_ID.....	101
VI_ATTR_RSRC_MANF_NAME .....	102
VI_ATTR_RSRC_NAME.....	102
VI_ATTR_RSRC_SPEC_VERSION .....	103

VI_ATTR_SEND_END_EN .....	104
VI_ATTR_SIGP_STATUS_ID.....	105
VI_ATTR_SLOT .....	106
VI_ATTR_SRC_ACCESS_PRIV .....	107
VI_ATTR_SRC_BYTE_ORDER .....	108
VI_ATTR_SRC_INCREMENT .....	109
VI_ATTR_STATUS .....	110
VI_ATTR_SUPPRESS_END_EN.....	111
VI_ATTR_TCPIP_ADDR .....	113
VI_ATTR_TCPIP_DEVICE_NAME .....	113
VI_ATTR_TCPIP_HOSTNAME .....	114
VI_ATTR_TCPIP_KEEPAIVE.....	115
VI_ATTR_TCPIP_NODELAY .....	116
VI_ATTR_TCPIP_PORT.....	116
VI_ATTR_TERMCHAR.....	117
VI_ATTR_TERMCHAR_EN .....	118
VI_ATTR_TMO_VALUE .....	119
VI_ATTR_TRIG_ID.....	120
VI_ATTR_USB_ALT_SETTING.....	122
VI_ATTR_USB_BULK_IN_PIPE.....	122
VI_ATTR_USB_BULK_IN_STATUS.....	123
VI_ATTR_USB_BULK_OUT_PIPE.....	124
VI_ATTR_USB_BULK_OUT_STATUS.....	125
VI_ATTR_USB_CLASS .....	126
VI_ATTR_USB_CTRL_PIPE .....	126
VI_ATTR_USB_END_IN .....	127
VI_ATTR_USB_INTFC_NUM .....	128
VI_ATTR_USB_INTR_IN_PIPE .....	129
VI_ATTR_USB_INTR_IN_STATUS .....	130
VI_ATTR_USB_MAX_INTR_SIZE .....	131
VI_ATTR_USB_NUM_INTFCS .....	132
VI_ATTR_USB_NUM_PIPES .....	132
VI_ATTR_USB_PROTOCOL .....	133
VI_ATTR_USB_RECV_INTR_DATA .....	134
VI_ATTR_USB_RECV_INTR_SIZE.....	135
VI_ATTR_USB_SERIAL_NUM .....	135

VI_ATTR_USB_SUBCLASS .....	136
VI_ATTR_USER_DATA/VI_ATTR_USER_DATA_32/VI_ATTR_USER_DATA_64 .....	137
VI_ATTR_VXI_DEV_CLASS .....	137
VI_ATTR_VXI_LA .....	138
VI_ATTR_VXI_TRIG_DIR .....	139
VI_ATTR_VXI_TRIG_LINES_EN .....	140
VI_ATTR_VXI_TRIG_STATUS .....	141
VI_ATTR_VXI_TRIG_SUPPORT .....	142
VI_ATTR_VXI_VME_INTR_STATUS .....	142
VI_ATTR_VXI_VME_SYSFAIL_STATE .....	143
VI_ATTR_WIN_ACCESS .....	144
VI_ATTR_WIN_ACCESS_PRIV .....	145
VI_ATTR_WIN_BASE_ADDR/VI_ATTR_WIN_BASE_ADDR_32/ VI_ATTR_WIN_BASE_ADDR_64 .....	147
VI_ATTR_WIN_BYTE_ORDER .....	148
VI_ATTR_WIN_SIZE/VI_ATTR_WIN_SIZE_32/VI_ATTR_WIN_SIZE_64 .....	149
VI_ATTR_WR_BUF_OPER_MODE .....	150
VI_ATTR_WR_BUF_SIZE .....	152
Events .....	153
VI_EVENT_ASRL_BREAK .....	153
VI_EVENT_ASRL_CHAR .....	154
VI_EVENT_ASRL_CTS .....	155
VI_EVENT_ASRL_DCD .....	155
VI_EVENT_ASRL_DSR .....	156
VI_EVENT_ASRL_RI .....	157
VI_EVENT_ASRL_TERMCHAR .....	158
VI_EVENT_CLEAR .....	159
VI_EVENT_EXCEPTION .....	160
VI_EVENT_GPIB_CIC .....	162
VI_EVENT_GPIB_LISTEN .....	162
VI_EVENT_GPIB_TALK .....	163
VI_EVENT_IO_COMPLETION .....	164
VI_EVENT_PXI_INTR .....	166
VI_EVENT_SERVICE_REQ .....	166
VI_EVENT_TRIG .....	167
VI_EVENT_USB_INTR .....	169

VI_EVENT_VXI_SIGP.....	170
VI_EVENT_VXI_VME_INTR .....	171
VI_EVENT_VXI_VME_SYSFAIL.....	172
VI_EVENT_VXI_VME_SYSRESET.....	172
Operations .....	174
viAssertIntrSignal.....	174
viAssertTrigger .....	177
viAssertUtilSignal.....	181
viBufRead.....	183
viBufWrite .....	186
viClear.....	189
viClose .....	192
viDisableEvent .....	193
viDiscardEvents .....	196
viEnableEvent.....	198
viEventHandler.....	201
viFindNext .....	203
viFindRsrc .....	205
viFlush .....	212
viGetAttribute .....	216
viGpibCommand .....	218
viGpibControlATN .....	221
viGpibControlREN .....	224
viGpibPassControl.....	227
viGpibSendIFC .....	229
viIn8/viIn16/viIn32/viIn64, viIn8Ex/viIn16Ex/viIn32Ex/viIn64Ex .....	231
viInstallHandler .....	235
viLock .....	238
viMapAddress/viMapAddressEx .....	242
viMapTrigger .....	246
viMemAlloc/viMemAllocEx.....	250
viMemFree/viMemFreeEx.....	253
viMove/viMoveEx .....	255
viMoveAsync/viMoveAsyncEx .....	260
viMoveIn8/viMoveIn16/viMoveIn32/viMoveIn64, viMoveIn8Ex/viMoveIn16Ex/ viMoveIn32Ex/viMoveIn64Ex .....	266

viMoveOut8/viMoveOut16/viMoveOut32/viMoveOut64, viMoveOut8Ex/viMoveOut16Ex/ viMoveOut32Ex/viMoveOut64Ex.....	271
viOpen .....	276
viOpenDefaultRM .....	280
viOut8/viOut16/viOut32/viOut64, viOut8Ex/viOut16Ex/viOut32Ex/viOut64Ex .....	283
viParseRsrc .....	287
viParseRsrcEx .....	290
viPeek8/viPeek16/viPeek32/viPeek64.....	294
viPoke8/viPoke16/viPoke32/viPoke64 .....	296
viPrintf .....	298
viQueryf .....	310
viRead.....	313
viReadAsync .....	317
viReadSTB .....	319
viReadToFile .....	323
viScanf .....	327
viSetAttribute .....	339
viSetBuf .....	342
viSPrintf .....	346
viSScanf .....	348
viStatusDesc .....	351
viTerminate .....	353
viUninstallHandler.....	355
viUnlock.....	357
viUnmapAddress .....	359
viUnmapTrigger .....	360
viUsbControlIn .....	364
viUsbControlOut .....	367
viVPrintf .....	370
viVQueryf .....	373
viVScanf .....	375
viVSPrintf.....	378
viVSScanf.....	380
viVxiCommandQuery .....	383
viWaitOnEvent .....	387
viWrite.....	390



viWriteAsync .....	393
viWriteFromFile .....	396
Resources .....	401
VISA Resource Template .....	401
VISA Resource Manager .....	402
INSTR Resource .....	403
MEMACC Resource .....	412
INTFC Resource .....	416
BACKPLANE Resource .....	419
SERVANT Resource .....	421
SOCKET Resource .....	423
RAW Resource .....	426
Error Codes .....	431
Completion Codes .....	436
NI-VISA Driver Wizard .....	438
Hardware Bus .....	438
Basic PXI/PCI Device Information .....	438
USB Device Selection .....	439
Basic USB Device Information .....	440
Interrupt Detection Information .....	441
Interrupt Removal Information .....	441
NI-VISA PXI Interrupt Information .....	442
Disarm Interrupt Information .....	443
PXI Express Configuration .....	443
Output Files Information .....	444
Installation Options .....	444

# NI-VISA API Reference

The following links take you to topics that describe the individual NI-VISA attributes, events, and operations. These are listed in alphabetical order within each access mechanism. Since a particular item can refer to more than one resource or interface type, each item is clearly marked with the resource and interface that support it.

Refer to [Resources](#) for a quick reference of how the attributes, events, and operations map to the available resources.

[Attributes](#)

[Events](#)

[Operations](#)

[VISA Access Mechanisms](#)

[VISA Resource Types](#)

# Attributes

These topics describe the VISA attributes. The attribute descriptions are listed in alphabetical order for easy reference.

Each attribute description contains a list below the title indicating the supported resource classes, such as GPIB, Serial, etc. The Attribute Information table lists the access privilege, the data type, range of values, and the default value.

## VI\_ATTR\_4882\_COMPLIANT

### Resource Classes

USB INSTR, VXI INSTR

### Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViBoolean	VI_TRUE (1) VI_FALSE (0)	N/A

### Description

VI\_ATTR\_4882\_COMPLIANT specifies whether the device is 488.2 compliant.

---

### Related Topics

[INSTR Resource](#)

## VI\_ATTR\_ASRL\_ALLOW\_TRANSMIT

# Resource Classes

Serial INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_TRUE

## Description

If set to VI\_FALSE, it suspends transmission as if an XOFF character has been received. If set to VI\_TRUE, it resumes transmission as if an XON character has been received.

If XON/XOFF flow control (software handshaking) is not being used, it is invalid to set this attribute to VI\_FALSE.

---

### Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_FLOW\\_CNTRL](#)

## VI\_ATTR\_ASRL\_AVAIL\_NUM

# Resource Classes

Serial INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
------------------	-----------	-------	---------

Read Only Global	ViUInt32	0 to FFFFFFFFh	N/A
---------------------	----------	----------------	-----

Description

VI\_ATTR\_ASRL\_AVAIL\_NUM shows the number of bytes available in the low-level I/O receive buffer.

---

Related Topics

[Controlling the Serial I/O Buffers](#)

[INSTR Resource](#)

VI\_ATTR\_ASRL\_BAUD

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViUInt32	0 to FFFFFFFFh	9600

Description

VI\_ATTR\_ASRL\_BAUD is the baud rate of the interface. It is represented as an unsigned 32-bit integer so that any baud rate can be used, but it usually requires a commonly used rate such as 300, 1200, 2400, or 9600 baud.

Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_DATA\\_BITS](#)

[VI\\_ATTR\\_ASRL\\_FLOW\\_CNTRL](#)

[VI\\_ATTR\\_ASRL\\_PARITY](#)

[VI\\_ATTR\\_ASRL\\_STOP\\_BITS](#)

VI\_ATTR\_ASRL\_BREAK\_LEN

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViInt16	1-500	250

Description

This controls the duration (in milliseconds) of the break signal asserted when `VI_ATTR_ASRL_END_OUT` is set to `VI_ATTR_ASRL_END_BREAK`. If you want to control the assertion state and length of a break signal manually, use the `VI_ATTR_ASRL_BREAK_STATE` attribute instead.

Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_BREAK\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_END\\_OUT](#)

VI\_ATTR\_ASRL\_BREAK\_STATE

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViInt16	VI_STATE_ASSERTED (1) VI_STATE_UNASSERTED (0) VI_STATE_UNKNOWN (-1)	VI_STATE_UNASSERTED

Description

If set to `VI_STATE_ASSERTED`, it suspends character transmission and places the transmission line in a break state until this attribute is reset to `VI_STATE_UNASSERTED`. This attribute lets you manually control the assertion state and length of a break signal. If you want VISA to send a break signal after each write operation automatically, use the `VI_ATTR_ASRL_BREAK_LEN` and `VI_ATTR_ASRL_END_OUT` attributes instead.

---

Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_BREAK\\_LEN](#)

VI\_ATTR\_ASRL\_END\_OUT

VI\_ATTR\_ASRL\_ALLOW\_TRANSMIT

VI\_ATTR\_ASRL\_CONNECTED

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViBoolean	VI_TRUE (1) VI_FALSE (0)	N/A

Description

VI\_ATTR\_ASRL\_CONNECTED indicates whether the port is properly connected to another port or device. This attribute is valid only with serial drivers developed by National Instruments and documented to support this feature with the corresponding National Instruments hardware.

---

Related Topics

[INSTR Resource](#)

VI\_ATTR\_ASRL\_CTS\_STATE

Resource Classes

Serial INSTR



# Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED (1) VI_STATE_UNASSERTED (0) VI_STATE_UNKNOWN (-1)	N/A

## Description

VI\_ATTR\_ASRL\_CTS\_STATE shows the current state of the Clear To Send (CTS) input signal.

---

### Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_DCD\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_DSR\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_DTR\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_RI\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_RTS\\_STATE](#)

## VI\_ATTR\_ASRL\_DATA\_BITS

### Resource Classes

Serial INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViUInt16	5 to 8	8

## Description

VI\_ATTR\_ASRL\_DATA\_BITS is the number of data bits contained in each frame (from 5 to 8). The data bits for each frame are located in the low-order bits of every byte stored in memory.

---

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_BAUD](#)

[VI\\_ATTR\\_ASRL\\_FLOW\\_CNTRL](#)

[VI\\_ATTR\\_ASRL\\_PARITY](#)

[VI\\_ATTR\\_ASRL\\_STOP\\_BITS](#)

## VI\_ATTR\_ASRL\_DCD\_STATE

## Resource Classes

Serial INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
------------------	-----------	-------	---------

Read/Write Global	ViInt16	VI_STATE_ASSERTED (1) VI_STATE_UNASSERTED (0) VI_STATE_UNKNOWN (-1)	N/A
----------------------	---------	---	-----

## Description

VI\_ATTR\_ASRL\_DCD\_STATE represents the current state of the Data Carrier Detect (DCD) input signal. The DCD signal is often used by modems to indicate the detection of a carrier (remote modem) on the telephone line. The DCD signal is also known as Receive Line Signal Detect (RLSD). This attribute is Read Only except when the VI\_ATTR\_ASRL\_WIRE\_MODE attribute is set to VI\_ASRL\_WIRE\_232\_DCE, or VI\_ASRL\_WIRE\_232\_AUTO with the hardware currently in the DCE state.

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_CTS\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_DSR\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_DTR\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_RI\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_RTS\\_STATE](#)

## VI\_ATTR\_ASRL\_DISCARD\_NULL



**Note** This attribute is supported for all serial ports on Windows and LabVIEW RT, and ENET-Serial on all platforms. Except for ENET-Serial, it is not supported for serial ports on Linux or Mac.

## Resource Classes

### Serial INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_FALSE

## Description

If set to `VI_TRUE`, NUL characters are discarded. Otherwise, they are treated as normal data characters. For binary transfers, set this attribute to `VI_FALSE`.

---

## Related Topics

[INSTR Resource](#)

## VI\_ATTR\_ASRL\_DSR\_STATE

## Resource Classes

### Serial INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED (1) VI_STATE_UNASSERTED (0) VI_STATE_UNKNOWN (-1)	N/A

## Description

`VI_ATTR_ASRL_DSR_STATE` shows the current state of the Data Set Ready (DSR) input signal.

---

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_CTS\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_DCD\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_DTR\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_RI\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_RTS\\_STATE](#)

## VI\_ATTR\_ASRL\_DTR\_STATE

## Resource Classes

Serial INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViInt16	VI_STATE_ASSERTED (1) VI_STATE_UNASSERTED (0) VI_STATE_UNKNOWN (-1)	N/A

## Description

`VI_ATTR_ASRL_DTR_STATE` shows the current state of the Data Terminal Ready (DTR) input signal.

When the `VI_ATTR_ASRL_FLOW_CNTRL` attribute is set to `VI_ASRL_FLOW_DTR_DSR`, this attribute is Read Only. Querying the value will return `VI_STATE_UNKNOWN`.

---

### Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_CTS\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_DCD\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_DSR\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_RI\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_RTS\\_STATE](#)

## VI\_ATTR\_ASRL\_END\_IN

### Resource Classes

Serial INSTR

### Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write	ViUInt16	VI_ASRL_END_NONE (0)	VI_ASRL_END_TERMCHAR

Local		VI_ASRL_END_LAST_BIT (1) VI_ASRL_END_TERMCHAR (2)	
-------	--	--	--

## Description

VI\_ATTR\_ASRL\_END\_IN indicates the method used to terminate read operations.

- If it is set to VI\_ASRL\_END\_NONE, the read will not terminate until all of the requested data is received (or an error occurs).
- If it is set to VI\_ASRL\_END\_LAST\_BIT, the read will terminate as soon as a character arrives with its last bit set. For example, if VI\_ATTR\_ASRL\_DATA\_BITS is set to 8, the read will terminate when a character arrives with the 8th bit set.
- If it is set to VI\_ASRL\_END\_TERMCHAR, the read will terminate as soon as the character in VI\_ATTR\_TERMCHAR is received. In this case, VI\_ATTR\_TERMCHAR\_LEN is ignored.

Because the default value of VI\_ATTR\_TERMCHAR is 0Ah (linefeed), read operations on serial ports will stop reading whenever a linefeed is encountered. To change this behavior, you must change the value of one of these attributes—VI\_ATTR\_ASRL\_END\_IN or VI\_ATTR\_TERMCHAR.

---

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_END\\_OUT](#)

[VI\\_ATTR\\_TERMCHAR](#)

## VI\_ATTR\_ASRL\_END\_OUT

## Resource Classes

### Serial INSTR

### Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_ASRL_END_NONE (0) VI_ASRL_END_LAST_BIT (1) VI_ASRL_END_TERMCHAR (2) VI_ASRL_END_BREAK (3)	VI_ASRL_END_NONE

### Description

`VI_ATTR_ASRL_END_OUT` indicates the method used to terminate write operations.

- If it is set to `VI_ASRL_END_NONE`, the write will transmit the exact contents of the user buffer, without modifying it and without appending anything to the data being written.
- If it is set to `VI_ASRL_END_LAST_BIT`, and `VI_ATTR_SEND_END_EN` is set to `VI_TRUE`, the write will send all but the last character with the highest bit clear, then transmit the last character with the highest bit set. For example, if `VI_ATTR_ASRL_DATA_BITS` is set to 8, the write will clear the eighth bit for all but the last character, then transmit the last character with the eighth bit set. If `VI_ATTR_SEND_END_EN` is set to `VI_FALSE`, the write will send all the characters with the highest bit clear.
- If it is set to `VI_ASRL_END_TERMCHAR`, and `VI_ATTR_SEND_END_EN` is set to `VI_TRUE`, the write will send the character in `VI_ATTR_TERMCHAR` after the data being transmitted. If `VI_ATTR_SEND_END_EN` is set to `VI_FALSE`, the write will transmit the exact contents of the user buffer, without modifying it and without appending anything to the data being written.
- If it is set to `VI_ASRL_END_BREAK`, and `VI_ATTR_SEND_END_EN` is set to `VI_TRUE`, the write will send the break character after the data being transmitted. If `VI_ATTR_SEND_END_EN` is set to `VI_FALSE`, the write will transmit the exact contents of the user buffer, without modifying it and without appending anything to the data being written.



I\_TRUE, the write will transmit a break after all the characters for the write have been sent. If VI\_ATTR\_SEND\_END\_EN is set to VI\_FALSE, the write will transmit the exact contents of the user buffer, without modifying it and without appending anything to the data being written.

Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_END\\_IN](#)

[VI\\_ATTR\\_TERMCHAR](#)

VI\_ATTR\_ASRL\_FLOW\_CNTRL

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViUInt16	VI_ASRL_FLOW_NONE (0) VI_ASRL_FLOW_XON_XOFF (1) VI_ASRL_FLOW_RTS_CTS (2) VI_ASRL_FLOW_DTR_DSR (4)	VI_ASRL_FLOW_NONE

Description

VI\_ATTR\_ASRL\_FLOW\_CNTRL indicates the type of flow control used by the

transfer mechanism.

- If this attribute is set to `VI_ASRL_FLOW_NONE`, the transfer mechanism does not use flow control, and buffers on both sides of the connection are assumed to be large enough to hold all data transferred.
- If this attribute is set to `VI_ASRL_FLOW_XON_XOFF`, the transfer mechanism uses the `XON` and `XOFF` characters to perform flow control. The transfer mechanism controls input flow by sending `XOFF` when the low-level I/O receive buffer is nearly full, and it controls the output flow by suspending transmission when `XOFF` is received.
- If this attribute is set to `VI_ASRL_FLOW_RTS_CTS`, the transfer mechanism uses the RTS output signal and the CTS input signal to perform flow control. The transfer mechanism controls input flow by unasserting the RTS signal when the low-level I/O receive buffer is nearly full, and it controls output flow by suspending the transmission when the CTS signal is unasserted.
- If this attribute is set to `VI_ASRL_FLOW_DTR_DSR`, the transfer mechanism uses the DTR output signal and the DSR input signal to perform flow control. The transfer mechanism controls input flow by unasserting the DTR signal when the low-level I/O receive buffer is nearly full, and it controls output flow by suspending the transmission when the DSR signal is unasserted.

This attribute can specify multiple flow control mechanisms by bit-ORing multiple values together. However, certain combinations may not be supported by all serial ports and/or operating systems.

---

## Related Topics

[Controlling the Serial I/O Buffers](#)

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_BAUD](#)

[VI\\_ATTR\\_ASRL\\_DATA\\_BITS](#)

[VI\\_ATTR\\_ASRL\\_PARITY](#)

[VI\\_ATTR\\_ASRL\\_STOP\\_BITS](#)

[VI\\_ATTR\\_ASRL\\_XOFF\\_CHAR](#)

[VI\\_ATTR\\_ASRL\\_XON\\_CHAR](#)

# VI\_ATTR\_ASRL\_PARITY

## Resource Classes

Serial INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViUInt16	VI_ASRL_PAR_NONE (0) VI_ASRL_PAR_ODD (1) VI_ASRL_PAR_EVEN (2) VI_ASRL_PAR_MARK (3) VI_ASRL_PAR_SPACE (4)	VI_ASRL_PAR_NONE

## Description

VI\_ATTR\_ASRL\_PARITY is the parity used with every frame transmitted and received.

- VI\_ASRL\_PAR\_MARK means that the parity bit exists and is always 1.
- VI\_ASRL\_PAR\_SPACE means that the parity bit exists and is always 0.

---

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_BAUD](#)

[VI\\_ATTR\\_ASRL\\_DATA\\_BITS](#)

[VI\\_ATTR\\_ASRL\\_FLOW\\_CNTRL](#)

[VI\\_ATTR\\_ASRL\\_STOP\\_BITS](#)

# VI\_ATTR\_ASRL\_REPLACE\_CHAR

## Resource Classes

Serial INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt8	0 to FFh	0

## Description

VI\_ATTR\_ASRL\_REPLACE\_CHAR specifies the character to be used to replace incoming characters that arrive with errors (such as parity error).

---

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_PARITY](#)

# VI\_ATTR\_ASRL\_RI\_STATE

## Resource Classes

### Serial INSTR

### Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViInt16	VI_STATE_ASSERTED (1) VI_STATE_UNASSERTED (0) VI_STATE_UNKNOWN (-1)	N/A

### Description

VI\_ATTR\_ASRL\_RI\_STATE represents the current state of the Ring Indicator (RI) input signal. The RI signal is often used by modems to indicate that the telephone line is ringing. This attribute is Read Only except when the VI\_ATTR\_ASRL\_WIRE\_MODE attribute is set to VI\_ASRL\_WIRE\_232\_DCE, or VI\_ASRL\_WIRE\_232\_AUTO with the hardware currently in the DCE state.

### Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_CTS\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_DCD\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_DSR\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_DTR\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_RTS\\_STATE](#)

## VI\_ATTR\_ASRL\_RTS\_STATE

## Resource Classes

### Serial INSTR

### Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViInt16	VI_STATE_ASSERTED (1) VI_STATE_UNASSERTED (0) VI_STATE_UNKNOWN (-1)	N/A

### Description

VI\_ATTR\_ASRL\_RTS\_STATE is used to manually assert or unassert the Request To Send (RTS) output signal.

When the VI\_ATTR\_ASRL\_FLOW\_CNTRL attribute is set to VI\_ASRL\_FLOW\_RTS\_CTS, this attribute is Read Only. Querying the value will return VI\_STATE\_UNKNOWN.

---

### Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_CTS\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_DCD\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_DSR\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_DTR\\_STATE](#)

[VI\\_ATTR\\_ASRL\\_RI\\_STATE](#)

# VI\_ATTR\_ASRL\_STOP\_BITS

## Resource Classes

Serial INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViUInt16	VI_ASRL_STOP_ONE (10) VI_ASRL_STOP_ONE5 (15) VI_ASRL_STOP_TWO (20)	VI_ASRL_STOP_ONE

## Description

VI\_ATTR\_ASRL\_STOP\_BITS is the number of stop bits used to indicate the end of a frame. The value VI\_ASRL\_STOP\_ONE5 indicates one-and-one-half (1.5) stop bits.

---

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_BAUD](#)

[VI\\_ATTR\\_ASRL\\_DATA\\_BITS](#)

[VI\\_ATTR\\_ASRL\\_FLOW\\_CNTRL](#)

[VI\\_ATTR\\_ASRL\\_PARITY](#)

# VI\_ATTR\_ASRL\_WIRE\_MODE

## Resource Classes

### Serial INSTR

### Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViInt16	VI_ASRL_WIRE_485_4 (0) VI_ASRL_WIRE_485_2_DTR_ECHO (1) VI_ASRL_WIRE_485_2_DTR_CTRL (2) VI_ASRL_WIRE_485_2_AUTO (3) VI_ASRL_WIRE_232_DTE (128) VI_ASRL_WIRE_232_DCE (129) VI_ASRL_WIRE_232_AUTO (130) V I_STATE_UNKNOWN (-1)	N/A

### Description

`VI_ATTR_ASRL_WIRE_MODE` represents the current wire/transceiver mode.

For RS-485 hardware, this attribute is valid only with the RS-485 serial driver developed by National Instruments.

For RS-232 hardware, the values RS232/DCE and RS232/AUTO are valid only with RS-232 serial drivers developed by National Instruments and documented to support this feature with the corresponding National Instruments hardware. When this feature is not supported, RS232/DTE is the only valid value.

RS-232 settings:

- `VI_ASRL_WIRE_232_DTE` uses DTE mode.
- `VI_ASRL_WIRE_232_DCE` uses DCE mode.
- `VI_ASRL_WIRE_232_AUTO` automatically detects which mode to use.



(Windows) RS-485 settings:

- `VI_ASRL_WIRE_485_4` uses 4-wire mode.
- `VI_ASRL_WIRE_485_2_DTR_ECHO` uses 2-wire DTR mode controlled with echo.
- `VI_ASRL_WIRE_485_2_DTR_CTRL` uses 2-wire DTR mode controlled without echo.
- `VI_ASRL_WIRE_485_2_AUTO` uses 2-wire auto mode controlled with TXRDY.

(Linux) RS-485 settings:

- `VI_ASRL_WIRE_485_4` uses 4-wire mode.
- `VI_ASRL_WIRE_485_2_AUTO` uses 2-wire auto mode controlled with TXRDY.



**Note** This attribute is valid only on the platforms on which National Instruments supports its RS-232 or RS-485 products.

Related Topics

[INSTR Resource](#)

VI\_ATTR\_ASRL\_XOFF\_CHAR

Resource Classes

Serial INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt8	0 to FFh	<Control-S> (13h)

## Description

`VI_ATTR_ASRL_XOFF_CHAR` specifies the value of the `XOFF` character used for `XON/XOFF` flow control (both directions). If `XON/XOFF` flow control (software handshaking) is not being used, the value of this attribute is ignored.

---

### Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_FLOW\\_CNTRL](#)

[VI\\_ATTR\\_ASRL\\_XON\\_CHAR](#)

## VI\_ATTR\_ASRL\_XON\_CHAR

### Resource Classes

Serial INSTR

### Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt8	0 to FFh	<Control-Q> (11h)

## Description

`VI_ATTR_ASRL_XON_CHAR` specifies the value of the `XON` character used for `XON/XOFF` flow control (both directions). If `XON/XOFF` flow control (software handshaking) is not being used, the value of this attribute is ignored.

---

---

**Related Topics**

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_FLOW\\_CNTRL](#)

[VI\\_ATTR\\_ASRL\\_XOFF\\_CHAR](#)

**VI\_ATTR\_BUFFER**

**Resource Classes**

[VI\\_EVENT\\_IO\\_COMPLETION](#)

**Attribute Information**

Access Privilege	Data Type	Range	Default
Read Only	ViBuf	N/A	N/A

**Description**

`VI_ATTR_BUFFER` contains the address of a buffer that was used in an asynchronous operation.

---

**Related Topics**

[VI\\_ATTR\\_JOB\\_ID](#)

[VI\\_ATTR\\_RET\\_COUNT/VI\\_ATTR\\_RET\\_COUNT\\_32/VI\\_ATTR\\_RET\\_COUNT\\_64](#)

[VI\\_ATTR\\_STATUS](#)

[VI\\_EVENT\\_IO\\_COMPLETION](#)

## VI\_ATTR\_CMDR\_LA

### Resource Classes

VXI INSTR, VXI SERVANT

### Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	0 to 255 VI_UNKNOWN_LA (-1)	N/A

### Description

VI\_ATTR\_CMDR\_LA is the unique logical address of the commander of the VXI device used by the given session.

---

### Related Topics

[INSTR Resource](#)

[SERVANT Resource](#)

## VI\_ATTR\_DEST\_ACCESS\_PRIV

### Resource Classes

VXI INSTR, VXI MEMACC


### Attribute Information

Access Privilege	Data Type	Range	Default
------------------	-----------	-------	---------

Read/Write Local	ViUInt16	VI_DATA_PRIV (0) VI_DATA_NPRIV (1) VI_PROG_PRIV (2) VI_PROG_NPRIV (3) VI_BLK_PRIV (4) VI_BLK_NPRIV (5) VI_D64_PRIV (6) VI_D64_NPRIV (7)	VI_DATA_PRIV
---------------------	----------	--	--------------

Description

VI\_ATTR\_DEST\_ACCESS\_PRIV specifies the address modifier to be used in high-level access operations, such as viOutXX() and viMoveOutXX(), when writing to the destination.



**Note** The values VI\_D64\_PRIV (6) and VI\_D64\_NPRIV (7) apply to only the block move operations. If you set this attribute to one of these values and then call one of the viOutXX() operations, the operation returns VI\_ERROR\_INV\_SETUP.

Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[VI\\_ATTR\\_DEST\\_BYTE\\_ORDER](#)

[VI\\_ATTR\\_DEST\\_INCREMENT](#)

[VI\\_ATTR\\_SRC\\_ACCESS\\_PRIV](#)

[VI\\_ATTR\\_WIN\\_ACCESS\\_PRIV](#)

VI\_ATTR\_DEST\_BYTE\_ORDER

# Resource Classes

VXI INSTR, VXI MEMACC

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_BIG_ENDIAN (0) VI_LITTLE_ENDIAN (1)	VI_BIG_ENDIAN

## Description

VI\_ATTR\_DEST\_BYTE\_ORDER specifies the byte order to be used in high-level access operations, such as viOutXX() and viMoveOutXX(), when writing to the destination.

---

### Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[VI\\_ATTR\\_DEST\\_ACCESS\\_PRIV](#)

[VI\\_ATTR\\_DEST\\_INCREMENT](#)

[VI\\_ATTR\\_SRC\\_BYTE\\_ORDER](#)

[VI\\_ATTR\\_WIN\\_BYTE\\_ORDER](#)

## VI\_ATTR\_DEST\_INCREMENT

## Resource Classes

PXI INSTR, PXI MEMACC, VXI INSTR, VXI MEMACC

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViInt32	0 to 1	1

## Description

`VI_ATTR_DEST_INCREMENT` is used in the `viMoveOutXX()` operations to specify by how many elements the destination offset is to be incremented after every transfer. The default value of this attribute is 1 (that is, the destination address will be incremented by 1 after each transfer), and the `viMoveOutXX()` operations move into consecutive elements. If this attribute is set to 0, the `viMoveOutXX()` operations will always write to the same element, essentially treating the destination as a FIFO register.

---

## Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[VI\\_ATTR\\_DEST\\_ACCESS\\_PRIV](#)

[VI\\_ATTR\\_DEST\\_BYTE\\_ORDER](#)

[VI\\_ATTR\\_SRC\\_INCREMENT](#)

`VI_ATTR_DEV_STATUS_BYTE`

## Resource Classes

GPIO INTFC, VXI SERVANT

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViUInt8	0 to FFh	N/A

## Description

This attribute specifies the 488-style status byte of the local controller or device associated with this session.

If this attribute is written and bit 6 (40h) is set, this device or controller will assert a service request (SRQ) if it is defined for this interface.

---

### Related Topics

[INTFC Resource](#)

[SERVANT Resource](#)

## VI\_ATTR\_DMA\_ALLOW\_EN

## Resource Classes

GPIO INSTR, GPIO INTFC, PXI INSTR, Serial INSTR, TCPIP INSTR, VXI INSTR, VXI MEMACC, VXI SERVANT



## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE (1) VI_FALSE (0)	N/A

## Description

This attribute specifies whether I/O accesses should use DMA (VI\_TRUE) or Programmed I/O (VI\_FALSE). In some implementations, this attribute may have global effects even though it is documented to be a local attribute. Since this affects performance and not functionality, that behavior is acceptable.

---

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[MEMACC Resource](#)

[SERVANT Resource](#)

## VI\_ATTR\_EVENT\_TYPE

### Resource Classes

All event object types

## Attribute Information

Access Privilege	Data Type	Range	Default
------------------	-----------	-------	---------

Read Only	ViEventType	0h to FFFFFFFFh	N/A
-----------	-------------	-----------------	-----

Description

VI\_ATTR\_EVENT\_TYPE is the unique logical identifier for the event type of the specified event.

---

Related Topics

[Events](#)

VI\_ATTR\_FDC\_CHNL

Resource Classes

VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	0 to 7	N/A

Description

VI\_ATTR\_FDC\_CHNL determines which Fast Data Channel (FDC) will be used to transfer the buffer.

---

Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_FDC\\_MODE](#)

[VI\\_ATTR\\_FDC\\_USE\\_PAIR](#)

VI\_ATTR\_FDC\_MODE

Resource Classes

VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_FDC_NORMAL (1) VI_FDC_STREAM (2)	VI_FDC_NORMAL

Description

VI\_ATTR\_FDC\_MODE specifies which Fast Data Channel (FDC) mode to use (either normal or stream mode).

---

Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_FDC\\_CHNL](#)

[VI\\_ATTR\\_FDC\\_USE\\_PAIR](#)

VI\_ATTR\_FDC\_USE\_PAIR

# Resource Classes

VXI INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_FALSE

## Description

Setting `VI_ATTR_FDC_USE_PAIR` to `VI_TRUE` specifies to use a channel pair for transferring data. Otherwise, only one channel will be used.

### Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_FDC\\_CHNL](#)

[VI\\_ATTR\\_FDC\\_MODE](#)

# VI\_ATTR\_FILE\_APPEND\_EN

## Resource Classes

GPIB INSTR, GPIB INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

## Attribute Information

Access Privilege	Data Type	Range	Default
------------------	-----------	-------	---------

Read/Write Local	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_FALSE
---------------------	-----------	-----------------------------	----------

Description

This attribute specifies whether `viReadToFile()` will overwrite (truncate) or append when opening a file.

---

Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[viReadToFile](#)

VI\_ATTR\_GPIB\_ADDR\_STATE

Resource Classes

GPIB INTFC

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_GPIB_UNADDRESSED (0) VI_GPIB_TALKER (1)	N/A

		VI_GPIB_LISTENER(2)	
--	--	---------------------	--

Description

This attribute shows whether the specified GPIB interface is currently addressed to talk or listen, or is not addressed.

Related Topics

[INTFC Resource](#)

[SERVANT Resource](#)

VI\_ATTR\_GPIB\_ATN\_STATE

Resource Classes

GPIB INTFC

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED(1) VI_STATE_UNASSERTED(0) VI_STATE_UNKNOWN(-1)	N/A

Description

This attribute shows the current state of the GPIB ATN (ATtention) interface line.

---

**Related Topics**

[INTFC Resource](#)

[VI\\_ATTR\\_GPIB\\_NDAC\\_STATE](#)

[VI\\_ATTR\\_GPIB\\_REN\\_STATE](#)

[VI\\_ATTR\\_GPIB\\_SRQ\\_STATE](#)

[viGpibControlATN](#)

**VI\_ATTR\_GPIB\_CIC\_STATE**

**Resource Classes**

GPIB INTFC

**Attribute Information**

Access Privilege	Data Type	Range	Default
Read Only Global	ViBoolean	VI_TRUE (1) VI_FALSE (0)	N/A

**Description**

This attribute shows whether the specified GPIB interface is currently CIC (Controller In Charge).

---

**Related Topics**

[INTFC Resource](#)

VI\_ATTR\_GPIB\_SYS\_CNTRL\_STATE

VI\_ATTR\_GPIB\_HS488\_CBL\_LEN

Resource Classes

GPIB INTFC

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViInt16	VI_GPIB_HS488_NIMPL(-1) VI_GPIB_HS488_DISABLE D(0) 1-15	N/A

Description

This attribute specifies the total number of meters of GPIB cable used in the specified GPIB interface.

---

Related Topics

[INTFC Resource](#)

[VI\\_ATTR\\_IO\\_PROT](#)

VI\_ATTR\_GPIB\_NDAC\_STATE

Resource Classes

GPIB INTFC



## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED (1) VI_STATE_UNASSERTED (0) VI_STATE_UNKNOWN (-1)	N/A

## Description

This attribute shows the current state of the GPIB NDAC (Not Data ACcepted) interface line.

---

## Related Topics

[INTFC Resource](#)

[VI\\_ATTR\\_GPIB\\_ATN\\_STATE](#)

[VI\\_ATTR\\_GPIB\\_REN\\_STATE](#)

[VI\\_ATTR\\_GPIB\\_SRQ\\_STATE](#)

## VI\_ATTR\_GPIB\_PRIMARY\_ADDR

## Resource Classes

GPIB INSTR, GPIB INTFC

## Attribute Information

Access Privilege	Data Type	Range	Default
------------------	-----------	-------	---------

INSTR, MEMACC, BACKPLANE: Read Only Global	ViUInt16	0 to 30	N/A
INTFC: Read/Write Global			

Description

VI\_ATTR\_GPIB\_PRIMARY\_ADDR specifies the primary address of the GPIB device used by the given session. For the GPIB INTFC Resource, this attribute is Read-Write.

Related Topics

[BACKPLANE Resource](#)

[INSTR Resource](#)

[INTFC Resource](#)

[MEMACC Resource](#)

[VI\\_ATTR\\_GPIB\\_SECONDARY\\_ADDR](#)

VI\_ATTR\_GPIB\_READADDR\_EN

Resource Classes

GPIB INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_TRUE

## Description

VI\_ATTR\_GPIB\_READDR\_EN specifies whether to use repeat addressing before each read or write operation.

---

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_GPIB\\_UNADDR\\_EN](#)

## VI\_ATTR\_GPIB\_RECV\_CIC\_STATE

## Resource Classes

VI\_EVENT\_GPIB\_CIC

## Attribute Information

Access Privilege	Data Type	Range	Default
Read-Only	ViBoolean	VI_TRUE (1) VI_FALSE (0)	N/A

## Description

This attribute specifies whether the local controller has gained or lost CIC status.

---

### Related Topics

[INTFC Resource](#)

[VI\\_ATTR\\_GPIB\\_ATN\\_STATE](#)

[VI\\_EVENT\\_GPIB\\_CIC](#)

## VI\_ATTR\_GPIB\_REN\_STATE

### Resource Classes

GPIB INSTR, GPIB INTFC

### Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED(1) VI_STATE_UNASSERTED(0) VI_STATE_UNKNOWN(-1)	N/A

## Description

**VI\_ATTR\_GPIB\_REN\_STATE** returns the current state of the GPIB REN (Remote ENable) interface line.

---

### Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[VI\\_ATTR\\_FILE\\_APPEND\\_EN](#)

[VI\\_ATTR\\_GPIB\\_ATN\\_STATE](#)

[VI\\_ATTR\\_GPIB\\_NDAC\\_STATE](#)

[VI\\_ATTR\\_GPIB\\_SRQ\\_STATE](#)

[viGpibControlREN](#)

# VI\_ATTR\_GPIB\_SECONDARY\_ADDR

## Resource Classes

GPIB INSTR, GPIB INTFC

## Attribute Information

Access Privilege	Data Type	Range	Default
INSTR, MEMACC, BACKPLANE: Read Only Global  INTFC: Read/Write Global	ViUInt16	0 to 30, VI_NO_SEC_ADDR (FFFFh)	N/A

## Description

VI\_ATTR\_GPIB\_SECONDARY\_ADDR specifies the secondary address of the GPIB device used by the given session. For the GPIB INTFC Resource, this attribute is Read-

Write.

---

**Related Topics**

[BACKPLANE Resource](#)

[INSTR Resource](#)

[INTFC Resource](#)

[MEMACC Resource](#)

[VI\\_ATTR\\_GPIB\\_PRIMARY\\_ADDR](#)

# VI\_ATTR\_GPIB\_SRQ\_STATE

## Resource Classes

GPIB INTFC

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED(1) VI_STATE_UNASSERTED(0) VI_STATE_UNKNOWN(-1)	N/A

## Description

This attribute shows the current state of the GPIB SRQ (Service ReQuest) interface line.

---

**Related Topics**

[INTFC Resource](#)

[VI\\_ATTR\\_GPIB\\_ATN\\_STATE](#)

[VI\\_ATTR\\_GPIB\\_NDAC\\_STATE](#)

[VI\\_ATTR\\_GPIB\\_REN\\_STATE](#)

VI\_ATTR\_GPIB\_SYS\_CNTRL\_STATE

Resource Classes

GPIB INTFC

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViBoolean	VI_TRUE(1) VI_FALSE(0)	N/A

Description

This attribute shows whether the specified GPIB interface is currently the system controller. In some implementations, this attribute may be modified only through a configuration utility. On these systems this attribute is read-only (RO).

Related Topics

[INTFC Resource](#)

[VI\\_ATTR\\_GPIB\\_CIC\\_STATE](#)

VI\_ATTR\_GPIB\_UNADDR\_EN

# Resource Classes

## GPIB INSTR

### Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_FALSE

### Description

VI\_ATTR\_GPIB\_UNADDR\_EN specifies whether to unaddress the device (UNT and UNL) after each read or write operation.

---

### Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_GPIB\\_READDR\\_EN](#)

## VI\_ATTR\_IMMEDIATE\_SERV

# Resource Classes

## VXI INSTR

### Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViBoolean	VI_TRUE (1)	N/A



Global		VI_FALSE (0)	
--------	--	--------------	--

Description

VI\_ATTR\_IMMEDIATE\_SERV specifies whether the device associated with this session is an immediate servant of the controller running VISA.

---

Related Topics

[INSTR Resource](#)

VI\_ATTR\_INTF\_INST\_NAME

Resource Classes


All I/O session types

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViString	N/A	N/A

Description

VI\_ATTR\_INTF\_INST\_NAME specifies human-readable text that describes the given interface.

**Note** The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

Related Topics

[BACKPLANE Resource](#)

[INSTR Resource](#)

[INTFC Resource](#)

[MEMACC Resource](#)

[SOCKET Resource](#)

[VI\\_ATTR\\_INTF\\_NUM](#)

[VI\\_ATTR\\_INTF\\_TYPE](#)

VI\_ATTR\_INTF\_NUM

Resource Classes

All I/O session types

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0h to FFFFh	0

Description

VI\_ATTR\_INTF\_NUM specifies the board number for the given interface.

Related Topics

[BACKPLANE Resource](#)

[INSTR Resource](#)

[INTFC Resource](#)

[MEMACC Resource](#)

[SOCKET Resource](#)

[VI\\_ATTR\\_INTF\\_TYPE](#)

# VI\_ATTR\_INTF\_TYPE

## Resource Classes

All I/O session types

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	VI_INTF_GPIB (1) VI_INTF_VXI (2) VI_INTF_GPIB_VXI (3) VI_INTF_ASRL (4) VI_INTF_PXI (5) VI_INTF_TCPIP (6) VI_INTF_USB (7)	N/A

## Description

VI\_ATTR\_INTF\_TYPE specifies the interface type of the given session.

---

## Related Topics

[BACKPLANE Resource](#)

[INSTR Resource](#)

[INTFC Resource](#)

[MEMACC Resource](#)

[SOCKET Resource](#)

[VI\\_ATTR\\_INTF\\_NUM](#)

# VI\_ATTR\_INTR\_STATUS\_ID

## Resource Classes

VI\_EVENT\_VXI\_VME\_INTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt32	0 to FFFFFFFFh	N/A

## Description

VI\_ATTR\_INTR\_STATUS\_ID specifies the 32-bit status/ID retrieved during the IACK cycle.

---

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

VI\_ATTR\_RECV\_INTR\_LEVELVI\_EVENT\_VXI\_VME\_INTR

## VI\_ATTR\_IO\_PROT

## Resource Classes

GPIO INTFC, GPIO INSTR, Serial INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	<b>GPIO:</b> VI_PROT_NORMAL (1) VI_PROT_HS488 (3)	VI_PROT_NORMAL
		<b>VXI</b> VI_PROT_NORMAL (1) VI_PROT_FDC (2)	VI_PROT_NORMAL
		<b>Serial, TCPIP, USB RAW:</b> VI_PROT_NORMAL (1) VI_PROT_4882_STRS (4)	VI_PROT_NORMAL
		<b>USB INSTR:</b> VI_PROT_NORMAL (1) VI_PROT_USBTMC_VENDOR (5)	VI_PROT_NORMAL

## Description

`VI_ATTR_IO_PROT` specifies which protocol to use. In VXI, you can choose normal word serial or fast data channel (FDC). In GPIB, you can choose normal or high-speed (HS-488) transfers. In serial, TCPIP, or USB RAW, you can choose normal transfers or 488.2-defined strings. In USB INSTR, you can choose normal or vendor-specific transfers.

In previous versions of VISA, `VI_PROT_NORMAL` was known as `VI_NORMAL`, `VI_PROT_FDC` was known as `VI_FDC`, `VI_PROT_HS488` was known as `VI_HS488`, and `VI_PROT_4882_STRS` was known as `VI_ASRL488`.

---

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[RAW Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[VI\\_ATTR\\_FDC\\_CHNL](#)

[VI\\_ATTR\\_FDC\\_MODE](#)

[VI\\_ATTR\\_FDC\\_USE\\_PAIR](#)

[VI\\_ATTR\\_GPIB\\_HS488\\_CBL\\_LEN](#)

## `VI_ATTR_JOB_ID`

# Resource Classes

VI\_EVENT\_IO\_COMPLETION

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViJobId	N/A	N/A

## Description

VI\_ATTR\_JOB\_ID contains the job ID of the asynchronous operation that has completed.

---

## Related Topics

[Resources](#)

[SERVANT Resource](#)

[VI\\_ATTR\\_BUFFER](#)

[VI\\_ATTR\\_RET\\_COUNT/VI\\_ATTR\\_RET\\_COUNT\\_32/VI\\_ATTR\\_RET\\_COUNT\\_64](#)

[VI\\_ATTR\\_STATUS](#)

[VI\\_EVENT\\_IO\\_COMPLETION](#)

# VI\_ATTR\_MAINFRAME\_LA

## Resource Classes

VXI INSTR, VXI BACKPLANE

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	0 to 255 VI_UNKNOWN_LA (-1)	N/A

## Description

VI\_ATTR\_MA.inFRAME\_LA specifies the lowest logical address in the mainframe. If the logical address is not known, VI\_UNKNOWN\_LA is returned.

---

## Related Topics

[BACKPLANE Resource](#)

[INSTR Resource](#)

## VI\_ATTR\_MANF\_ID

## Resource Classes

PXI INSTR, USB INSTR, USB RAW, VXI INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0h to FFFFh	N/A



# Description

VI\_ATTR\_MANF\_ID is the manufacturer identification number of the device.

For VXI resources, this refers to the VXI Manufacturer ID.

For PXI INSTR resources, if the subsystem PCI Vendor ID is nonzero, this refers to the subsystem Vendor ID. Otherwise, this refers to the Vendor ID.

For USB resources, this refers to the Vendor ID (VID).

---

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_MANF\\_NAME](#)

[VI\\_ATTR\\_MODEL\\_CODE](#)

# VI\_ATTR\_MANF\_NAME

## Resource Classes

PXI INSTR, PXI BACKPLANE, USB INSTR, USB RAW, VXI INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViString	N/A	N/A

## Description

This string attribute is the manufacturer name.



**Note** The value of this attribute should be used for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

---

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_MANF\\_ID](#)

[VI\\_ATTR\\_MODEL\\_NAME](#)

# VI\_ATTR\_MAX\_QUEUE\_LENGTH

## Resource Classes

All I/O session types

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt32	1h to FFFFFFFFh	50

## Description

`VI_ATTR_MAX_QUEUE_LENGTH` specifies the maximum number of events that can be queued at any time on the given session. Events that occur after the queue has become full will be discarded.

`VI_ATTR_MAX_QUEUE_LENGTH` is a Read/Write attribute until the first time `viEna`

`bleEvent()` is called on a session. Thereafter, this attribute is Read Only.

Related Topics

[Operations](#)

[viEnableEvent](#)

[VISA Resource Template](#)

[viWaitOnEvent](#)

VI\_ATTR\_MEM\_BASE/VI\_ATTR\_MEM\_BASE\_32/  
VI\_ATTR\_MEM\_BASE\_64

Resource Classes

VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	<b>VI_ATTR_MEM_BASE:</b> ViBusAddress  <b>VI_ATTR_MEM_BASE_32:</b> ViUInt32  <b>VI_ATTR_MEM_BASE_64:</b> ViUInt64	<b>VI_ATTR_MEM_BASE:</b> 0h to FFFFFFFFh for 32-bit applications  0h to FFFFFFFFFFFFFFFFh for 64-bit applications  <b>VI_ATTR_MEM_BASE_32:</b> 0h to FFFFFFFFh  <b>VI_ATTR_MEM_BASE_64:</b>	N/A

		0h to FFFFFFFFFFFFFFFFh	
--	--	-------------------------	--

Description

VI\_ATTR\_MEM\_BASE, VI\_ATTR\_MEM\_BASE\_32, and VI\_ATTR\_MEM\_BASE\_64 specify the base address of the device in VXIbus memory address space. This base address is applicable to A24 or A32 address space. If the value of VI\_ATTR\_MEM\_SPACE is VI\_A16\_SPACE, the value of this attribute is meaningless for the given VXI device.

Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_MEM\\_SIZE/VI\\_ATTR\\_MEM\\_SIZE\\_32/VI\\_ATTR\\_MEM\\_SIZE\\_64](#)

[VI\\_ATTR\\_MEM\\_SPACE](#)

VI\_ATTR\_MEM\_SIZE/VI\_ATTR\_MEM\_SIZE\_32/  
VI\_ATTR\_MEM\_SIZE\_64

Resource Classes

VXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	VI_ATTR_MEM_SIZE: ViBusSize	VI_ATTR_MEM_SIZE: 0h to FFFFFFFFh for 32-bit applications	N/A

Global	<b>VI_ATTR_MEM_SIZE_32:</b> ViUInt32  <b>VI_ATTR_MEM_SIZE_64:</b> ViUInt64	0h to FFFFFFFFFFFFFFFFh for 64-bit applications  <b>VI_ATTR_MEM_SIZE_32:</b> 0h to FFFFFFFFh  <b>VI_ATTR_MEM_SIZE_64:</b> 0h to FFFFFFFFFFFFFFFFh	
--------	--	---	--

Description

VI\_ATTR\_MEM\_SIZE, VI\_ATTR\_MEM\_SIZE\_32, and VI\_ATTR\_MEM\_SIZE\_64 specify the size of memory requested by the device in VXIbus address space. If the value of VI\_ATTR\_MEM\_SPACE is VI\_A16\_SPACE, the value of this attribute is meaningless for the given VXI device.

Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_MEM\\_BASE/VI\\_ATTR\\_MEM\\_BASE\\_32/VI\\_ATTR\\_MEM\\_BASE\\_64](#)

[VI\\_ATTR\\_MEM\\_SPACE](#)

VI\_ATTR\_MEM\_SPACE

Resource Classes

VXI INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	VI_A16_SPACE (1) VI_A24_SPACE (2) VI_A32_SPACE (3)	VI_A16_SPACE

## Description

`VI_ATTR_MEM_SPACE` specifies the VXIbus address space used by the device. The three types are A16, A24, or A32 memory address space.

A VXI device with memory in A24 or A32 space also has registers accessible in the configuration section of A16 space. A VME device with memory in multiple address spaces requires one VISA resource for each address space used.

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_MEM\\_BASE/VI\\_ATTR\\_MEM\\_BASE\\_32/VI\\_ATTR\\_MEM\\_BASE\\_64](#)

[VI\\_ATTR\\_MEM\\_SIZE/VI\\_ATTR\\_MEM\\_SIZE\\_32/VI\\_ATTR\\_MEM\\_SIZE\\_64](#)

## VI\_ATTR\_MODEL\_CODE

## Resource Classes

PXI INSTR, USB INSTR, USB RAW, VXI INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
------------------	-----------	-------	---------

Read Only Global	ViUInt16	0h to FFFFh	N/A
---------------------	----------	-------------	-----

## Description

`VI_ATTR_MODEL_CODE` specifies the model code for the device.

For VXI resources, this refers to the VXI Model Code.

For PXI INSTR resources, if the subsystem PCI Vendor ID is nonzero, this refers to the subsystem Device ID. Otherwise, this refers to the Device ID.

For USB resources, this refers to the Product ID (PID).

---

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_MANF\\_ID](#)

[VI\\_ATTR\\_MODEL\\_NAME](#)

## VI\_ATTR\_MODEL\_NAME

## Resource Classes

PXI INSTR, PXI BACKPLANE, USB INSTR, USB RAW, VXI INSTR


## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViString	N/A	N/A

Global			
--------	--	--	--

Description

This string attribute is the model name of the device.



**Note** The value of this attribute should be used for display purposes only and not for programmatic decisions, as the value can be different between VISA implementations and/or revisions.

Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_MANF\\_NAME](#)

[VI\\_ATTR\\_MODEL\\_CODE](#)

VI\_ATTR\_OPER\_NAME

Resource Classes

VI\_EVENT\_IO\_COMPLETION, VI\_EVENT\_EXCEPTION

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViString	N/A	N/A



# Description

VI\_ATTR\_OPER\_NAME contains the name of the operation generating this event.

---

## Related Topics

[VI\\_ATTR\\_EVENT\\_TYPE](#)

[VI\\_ATTR\\_STATUS](#)

[VI\\_EVENT\\_EXCEPTION](#)

[VI\\_EVENT\\_IO\\_COMPLETION](#)

# VI\_ATTR\_PXI\_ACTUAL\_LWIDTH

## Resource Classes

PXI INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	-1, 1, 2, 4, 8, 16	N/A

# Description

VI\_ATTR\_PXI\_ACTUAL\_LWIDTH specifies the PCI Express link width negotiated between the PCI Express host controller and the device. A value of -1 indicates that the device is not a PXI/PCI Express device.

---

## Related Topics

[INSTR Resource](#)

# VI\_ATTR\_PXI\_BUS\_NUM

## Resource Classes

PXI INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0 to 255	N/A

## Description

VI\_ATTR\_PXI\_BUS\_NUM specifies the PCI bus number of this device.

---

## Related Topics

[INSTR Resource](#)

# VI\_ATTR\_PXI\_CHASSIS

## Resource Classes

PXI INSTR, PXI BACKPLANE

## Attribute Information

Access Privilege	Data Type	Range	Default
------------------	-----------	-------	---------

Read Only Global	ViInt16	-1, 0 to 255	N/A
------------------	---------	--------------	-----

Description

VI\_ATTR\_PXI\_CHASSIS specifies the PXI chassis number of this device. A value of -1 means the chassis number is unknown.

Related Topics

[INSTR Resource](#)

VI\_ATTR\_PXI\_DEST\_TRIG\_BUS

Resource Classes

PXI BACKPLANE

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViInt16	<b>Single-Segment Chassis (8 Slots or Less):</b> N/A  <b>Multisegment Chassis (More than 8 Slots):</b> 1...number of chassis segments*	-1

Description

VI\_ATTR\_PXI\_DEST\_TRIG\_BUS specifies the segment to use to qualify trigDes t in viMapTrigger.



**Note** Some PXI chassis, typically those with more than 8 slots, have multiple trigger buses (also called segments). `viMapTrigger` is used on the PXI BACKPLANE resource to map a trigger between two trigger buses. One trigger bus, specified by `VI_ATTR_PXI_SRC_TRIG_BUS`, is the source or "writer" for this trigger line. The other trigger bus, specified by `VI_ATTR_PXI_DEST_TRIG_BUS`, is a "reader." You can have multiple readers, but only one writer for a given trigger line. For example, if you want to have triggers mapped from trigger bus 1 to trigger bus 2 and then from trigger bus 2 to trigger bus 3, observe that in this case trigger bus 1 is the writer for this line, writing to both trigger bus 2 and trigger bus 3. Therefore, you should perform your `viMapTrigger` from 1 to 2 and from 1 to 3—mapping from 1 to 2 and then 2 to 3 would not be allowed because it would require 2 also to be a writer (as well as 1). Note also that mapping from one line in the source trigger bus to a different line in the destination trigger bus (`trigSrc != trigDest`) is dependent on hardware capabilities and a specific software implementation, and may not be supported.

Code to map trigger 5 from trigger segment 1 to trigger segment 2 of an 18-slot chassis would look like the following, where `backplaneSession` is a session to a PXI BACKPLANE resource:

```
viSetAttribute(backplaneSession, VI_ATTR_PXI_SRC_TRIG_BUS, 1);
viSetAttribute(backplaneSession, VI_ATTR_PXI_DEST_TRIG_BUS, 2);
viMapTrigger(backplaneSession, VI_TRIG_TTL5, VI_TRIG_TTL5, VI_NULL);
```

\*You can determine the number of segments from MAX (in the trigger reservation panel), from the chassis documentation, and by looking at the dividing lines on the physical front panel of the chassis itself.

## Related Topics

[BACKPLANE Resource](#)

[VI\\_ATTR\\_PXI\\_SRC\\_TRIG\\_BUS](#)

[VI\\_ATTR\\_PXI\\_TRIG\\_BUS](#)

[viMapTrigger](#)

[VI\\_ATTR\\_PXI\\_DEV\\_NUM](#)

## Resource Classes

PXI INSTR

### Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0 to 31	N/A

### Description

This is the PXI device number.

---

### Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_PXI\\_FUNC\\_NUM](#)

## VI\_ATTR\_PXI\_DSTAR\_BUS

## Resource Classes

PXI INSTR

### Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	N/A	N/A

## Description

`VI_ATTR_PXI_DSTAR_BUS` specifies the differential star bus number of this device. A value of `-1` means the chassis is unidentified or does not have a timing slot.

---

### Related Topics

[INSTR Resource](#)

## VI\_ATTR\_PXI\_DSTAR\_SET

### Resource Classes

PXI INSTR

### Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	-1, 0 to 16	N/A

## Description

`VI_ATTR_PXI_DSTAR_SET` specifies the set of PXIe DStar lines connected to the slot this device is in. Each slot can be connected to a set of DStar lines, and each set has a number. For example, one slot could be connected to the DStar set 2, while the next one could be connected to the DStar set 4. The `VI_ATTR_PXI_DSTAR_SET` value does not represent individual line numbers; instead, it represents the number of the set itself. A PXIe DStar set consists of the numbered differential pairs PXIe-DSTARA, PXIe-DSTARB, and PXIe-DSTARC routed from the PXIe system timing slot. For example, if `VI_ATTR_PXI_DSTAR_SET` is 4, the slot the device is in is connected to PXIe-DStarA\_4, PXIe-DStarB\_4, and PXIe-DStarC\_4. A value of `-1` means the chassis is unidentified or the slot the device is in does not have a DStar set connected to it.

Also, although a PXIe slot has a DStar connection, the device in that slot may not. In that case, the value of `VI_ATTR_PXI_DSTAR_SET` still will be the set connected to the slot the device is in.

---

**Related Topics**

[INSTR Resource](#)

# VI\_ATTR\_PXI\_FUNC\_NUM

## Resource Classes

PXI INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0 to 7	0

## Description

This is the PCI function number of the PXI/PCI resource. For most devices, the function number is 0, but a multifunction device may have a function number up to 7. The meaning of a function number other than 0 is device specific.

---

**Related Topics**

[INSTR Resource](#)

[VI\\_ATTR\\_PXI\\_DEV\\_NUM](#)

# VI\_ATTR\_PXI\_IS\_EXPRESS

## Resource Classes

PXI INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViBoolean	VI_TRUE, VI_FALSE	N/A

## Description

VI\_ATTR\_PXI\_IS\_EXPRESS specifies whether the device is PXI/PCI or PXI/PCI Express.

---

## Related Topics

[INSTR Resource](#)

# VI\_ATTR\_PXI\_MAX\_LWIDTH

## Resource Classes

PXI INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	-1, 1, 2, 4, 8, 16	N/A



## Description

`VI_ATTR_PXI_MAX_LWIDTH` specifies the maximum PCI Express link width of the device. A value of -1 indicates that the device is not a PXI/PCI Express device.

---

### Related Topics

[INSTR Resource](#)

`VI_ATTR_PXI_MEM_BASE_BAR0/`  
`VI_ATTR_PXI_MEM_BASE_BAR1/`  
`VI_ATTR_PXI_MEM_BASE_BAR2/`  
`VI_ATTR_PXI_MEM_BASE_BAR3/`  
`VI_ATTR_PXI_MEM_BASE_BAR4/`  
`VI_ATTR_PXI_MEM_BASE_BAR5`

## Resource Classes

PXI INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt32	0 to FFFFFFFFh	N/A

## Description

PXI memory base address assigned to the specified BAR. If the value of the corresponding `VI_ATTR_PXI_MEM_TYPE_BARx` is `VI_PXI_ADDR_NONE`, the value of this attribute is undefined for the given PXI device.

Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_PXI\\_MEM\\_SIZE\\_BAR0/1/2/3/4/5](#)

[VI\\_ATTR\\_PXI\\_MEM\\_TYPE\\_BAR0/1/2/3/4/5](#)

VI\_ATTR\_PXI\_MEM\_SIZE\_BAR0/  
VI\_ATTR\_PXI\_MEM\_SIZE\_BAR1/  
VI\_ATTR\_PXI\_MEM\_SIZE\_BAR2/  
VI\_ATTR\_PXI\_MEM\_SIZE\_BAR3/  
VI\_ATTR\_PXI\_MEM\_SIZE\_BAR4/  
VI\_ATTR\_PXI\_MEM\_SIZE\_BAR5

Resource Classes

PXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt32	0 to FFFFFFFFh	N/A

Description

Memory size used by the device in the specified BAR. If the value of the corresponding VI\_ATTR\_PXI\_MEM\_TYPE\_BAR*x* is VI\_PXI\_ADDR\_NONE, the value of this attribute is undefined for the given PXI device.

Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_PXI\\_MEM\\_BASE\\_BAR0/1/2/3/4/5](#)

[VI\\_ATTR\\_PXI\\_MEM\\_TYPE\\_BAR0/1/2/3/4/5](#)

VI\_ATTR\_PXI\_MEM\_TYPE\_BAR0/  
VI\_ATTR\_PXI\_MEM\_TYPE\_BAR1/  
VI\_ATTR\_PXI\_MEM\_TYPE\_BAR2/  
VI\_ATTR\_PXI\_MEM\_TYPE\_BAR3/  
VI\_ATTR\_PXI\_MEM\_TYPE\_BAR4/  
VI\_ATTR\_PXI\_MEM\_TYPE\_BAR5

Resource Classes

PXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	VI_PXI_ADDR_NONE (0) VI_PXI_ADDR_MEM (1) VI_PXI_ADDR_IO (2)	N/A

Description

Memory type used by the device in the specified BAR (if applicable).

---

**Related Topics**

[INSTR Resource](#)

[VI\\_ATTR\\_PXI\\_MEM\\_BASE\\_BAR0/1/2/3/4/5](#)

[VI\\_ATTR\\_PXI\\_MEM\\_SIZE\\_BAR0/1/2/3/4/5](#)

**VI\_ATTR\_PXI\_RECV\_INTR\_DATA**

**Resource Classes**

**VI\_EVENT\_PXI\_INTR**

**Attribute Information**

Access Privilege	Data Type	Range	Default
Read Only	ViUInt32	N/A	N/A

**Description**

VI\_ATTR\_PXI\_RECV\_INTR\_DATA shows the first PXI/PCI register that was read in the successful interrupt detection sequence.

---

**Related Topics**

[VI\\_ATTR\\_PXI\\_RECV\\_INTR\\_SEQ](#)

[VI\\_EVENT\\_PXI\\_INTR](#)

**VI\_ATTR\_PXI\_RECV\_INTR\_SEQ**

## Resource Classes

VI\_EVENT\_PXI\_INTR

### Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViInt16	N/A	N/A

### Description

VI\_ATTR\_PXI\_RECV\_INTR\_SEQ shows the index of the interrupt sequence that detected the interrupt condition.

---

### Related Topics

[VI\\_ATTR\\_PXI\\_RECV\\_INTR\\_DATA](#)

[VI\\_EVENT\\_PXI\\_INTR](#)

## VI\_ATTR\_PXI\_SLOT\_LBUS\_LEFT

### Resource Classes

PXI INSTR

### Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViInt16	VI_PXI_LBUS_UNKNOWN (-1);	N/A

Global		<pre>VI_PXI_LBUS_NONE (0); Normal slots (1 to 18); VI_PXI_LBUS_STAR_TRIG_BUS_0 (1000) to V I_PXI_LBUS_STAR_TRIG_BUS_9 (1009); VI_PXI_STAR_TRIG_CONTROLLER (1413); VI_PXI_LBUS_SCXI (2000)</pre>	
--------	--	---	--

Description

VI\_ATTR\_PXI\_SLOT\_LBUS\_LEFT specifies the slot number or special feature connected to the local bus left lines of this device.

---

Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_PXI\\_SLOT\\_LBUS\\_RIGHT](#)

[VI\\_ATTR\\_SLOT](#)

VI\_ATTR\_PXI\_SLOT\_LBUS\_RIGHT

Resource Classes

PXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	<pre>VI_PXI_LBUS_UNKNOWN (-1); VI_PXI_LBUS_NONE (0);</pre>	N/A

		Normal slots (1 to 18); VI_PXI_LBUS_STAR_TRIG_BUS_0 (1000) to V I_PXI_LBUS_STAR_TRIG_BUS_9 (1009); VI_PXI_STAR_TRIG_CONTROLLER (1413); VI_PXI_LBUS_SCXI (2000)	
--	--	--	--

Description

VI\_ATTR\_PXI\_SLOT\_LBUS\_RIGHT specifies the slot number or special feature connected to the local bus right lines of this device.

Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_PXI\\_SLOT\\_LBUS\\_LEFT](#)

[VI\\_ATTR\\_SLOT](#)

VI\_ATTR\_PXI\_SLOT\_LWIDTH

Resource Classes

PXI INSTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	-1, 1, 4, 8	N/A

# Description

`VI_ATTR_PXI_SLOT_LWIDTH` specifies the PCI Express link width of the PXI Express peripheral slot in which the device resides. A value of -1 indicates that the device is not a PXI Express device.

---

## Related Topics

[INSTR Resource](#)

# VI\_ATTR\_PXI\_SLOTPATH

## Resource Classes

PXI INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViString	N/A	N/A

# Description

`VI_ATTR_PXI_SLOTPATH` specifies the slot path of this device.

The purpose of a PXI slot path is to describe the PCI bus hierarchy in a manner independent of the PCI bus number. PXI slot paths are a sequence of values representing the PCI device number and function number of a PCI module and each parent PCI bridge that routes the module to the host PCI bridge (bus 0). Each value is represented as "dev[.func]", where the function number is listed only if it is non-zero. When a PXI slot path includes multiple values, the values are comma-separated.

The string format of the attribute value looks like this:



```
device1[.function1][,device2[.function2]][,...]
```

An example string is "5.1,12,8". In this case, there is a PCI-to-PCI bridge on device 8 on the root bus. On its secondary bus, there is another PCI-to-PCI bridge on device 12. On its secondary bus, there is an instrument on device 5, function 1. The example string value describes this instrument's slot path.

Related Topics

[INSTR Resource](#)

VI\_ATTR\_PXI\_SRC\_TRIG\_BUS

Resource Classes

PXI BACKPLANE

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViInt16	Single-Segment Chassis (8 Slots or Less): N/A  Multisegment Chassis (More than 8 Slots): 1...number of chassis segments*	-1

Description

VI\_ATTR\_PXI\_SRC\_TRIG\_BUS specifies the segment to use to qualify `trigSrc` in `viMapTrigger`.

**Note** Some PXI chassis, typically those with more than 8 slots, have multiple trigger buses (also called segments). `viMapTrigger` is used on the PXI BACKPLANE resource to map a trigger between two trigger buses. One trigger bus, specified by `VI_ATTR_PXI_SRC_TRIG_BUS`, is the source or "writer" for this trigger line. The other trigger bus, specified by `VI_ATTR_PXI_DEST_TRIG_BUS`, is a "reader." You can have multiple readers, but only one writer for a given trigger line. For example, if you want to have triggers mapped from trigger bus 1 to trigger bus 2 and then from trigger bus 2 to trigger bus 3, observe that in this case trigger bus 1 is the writer for this line, writing to both trigger bus 2 and trigger bus 3. Therefore, you should perform your `viMapTrigger` from 1 to 2 and from 1 to 3—mapping from 1 to 2 and then 2 to 3 would not be allowed because it would require 2 also to be a writer (as well as 1). Note also that mapping from one line in the source trigger bus to a different line in the destination trigger bus (`trigSrc != trigDest`) is dependent on hardware capabilities and a specific software implementation, and may not be supported.



Code to map trigger 5 from trigger segment 1 to trigger segment 2 of an 18-slot chassis would look like the following, where `backplaneSession` is a session to a PXI BACKPLANE resource:

```
viSetAttribute(backplaneSession, VI_ATTR_PXI_SRC_TRIG_BUS, 1);
viSetAttribute(backplaneSession, VI_ATTR_PXI_DEST_TRIG_BUS, 2);
viMapTrigger(backplaneSession, VI_TRIG_TTL5, VI_TRIG_TTL5, VI_NULL);
```

\*You can determine the number of segments from MAX (in the trigger reservation panel), from the chassis documentation, and by looking at the dividing lines on the physical front panel of the chassis itself.

## Related Topics

[BACKPLANE Resource](#)

[VI\\_ATTR\\_PXI\\_DEST\\_TRIG\\_BUS](#)

[VI\\_ATTR\\_PXI\\_TRIG\\_BUS](#)

[viMapTrigger](#)

[VI\\_ATTR\\_PXI\\_STAR\\_TRIG\\_BUS](#)

## Resource Classes

PXI INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	N/A	N/A

## Description

`VI_ATTR_PXI_STAR_TRIG_BUS` specifies the star trigger bus number of this device.

---

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_PXI\\_STAR\\_TRIG\\_LINE](#)

[VI\\_ATTR\\_PXI\\_TRIG\\_BUS](#)

## VI\_ATTR\_PXI\_STAR\_TRIG\_LINE

## Resource Classes

PXI INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
------------------	-----------	-------	---------

Read Only Global	ViInt16	N/A	N/A
------------------	---------	-----	-----

Description

VI\_ATTR\_PXI\_STAR\_TRIG\_LINE specifies the PXI\_STAR line connected to this device.

Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_PXI\\_STAR\\_TRIG\\_BUS](#)

[VI\\_ATTR\\_PXI\\_TRIG\\_BUS](#)

VI\_ATTR\_PXI\_TRIG\_BUS

Resource Classes

PXI INSTR, PXI BACKPLANE

Attribute Information

Access Privilege	Data Type	Range	Default
INSTR: Read Only Global  BACKPLANE: Read/Write Local	ViInt16	N/A	N/A

## Description

`VI_ATTR_PXI_TRIG_BUS` specifies the trigger bus number of this device.

---

### Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_PXI\\_STAR\\_TRIG\\_BUS](#)

[VI\\_ATTR\\_PXI\\_STAR\\_TRIG\\_LINE](#)

## VI\_ATTR\_RD\_BUF\_OPER\_MODE

### Resource Classes

GPIO INSTR, GPIB INTRFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

### Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_FLUSH_ON_ACCESS (1) VI_FLUSH_DISABLE (3)	VI_FLUSH_DISABLE

## Description

`VI_ATTR_RD_BUF_OPER_MODE` specifies the operational mode of the formatted I/O read buffer. When the operational mode is set to `VI_FLUSH_DISABLE` (default), the buffer is flushed only on explicit calls to `viFlush()`. If the operational mode is set to `VI_FLUSH_ON_ACCESS`, the read buffer is flushed every time a `viScanf()` (or related) operation completes.

Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[VI\\_ATTR\\_WR\\_BUF\\_OPER\\_MODE](#)

[viFlush](#)

[viScanf](#)

VI\_ATTR\_RD\_BUF\_SIZE

Resource Classes

GPIB INSTR, GPIB INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Local	ViUInt32	N/A	N/A

Description

This is the current size of the formatted I/O input buffer for this session. The user can modify this value by calling `viSetBuf()`.

Related Topics

[VI\\_ATTR\\_RD\\_BUF\\_OPER\\_MODE](#)

[VI\\_ATTR\\_WR\\_BUF\\_SIZE](#)

[viSetBuf](#)

VI\_ATTR\_RECV\_INTR\_LEVEL

Resource Classes

VI\_EVENT\_VXI\_VME\_INTR

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViInt16	1 to 7; VI_UNKNOWN_LEVEL (-1)	N/A

Description

VI\_ATTR\_RECV\_INTR\_LEVEL is the VXI interrupt level on which the interrupt was received.

Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

[VI\\_ATTR\\_INTR\\_STATUS\\_ID](#)

VI\_EVENT\_VXI\_VME\_INTR

VI\_ATTR\_RECV\_TRIG\_ID

Resource Classes

VI\_EVENT\_TRIG

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViInt16	VI_TRIG_SW(-1) VI_TRIG_TTL0 (0) to VI_TRIG_TTL7 (7); VI_TRIG_ECL0 (8) to VI_TRIG_ECL1 (9)	N/A

Description

VI\_ATTR\_RECV\_TRIG\_ID identifies the triggering mechanism on which the specified trigger event was received.

Related Topics

[BACKPLANE Resource](#)

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[VI\\_EVENT\\_TRIG](#)



# VI\_ATTR\_RET\_COUNT/VI\_ATTR\_RET\_COUNT\_32/ VI\_ATTR\_RET\_COUNT\_64

## Resource Classes

VI\_EVENT\_IO\_COMPLETION

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	<b>VI_ATTR_RET_COUNT:</b> ViUInt32 for 32-bit applications  ViUInt64 for 64-bit applications  <b>VI_ATTR_RET_COUNT_32:</b> ViUInt32  <b>VI_ATTR_RET_COUNT_64:</b> ViUInt64	<b>VI_ATTR_RET_COUNT:</b> 0h to FFFFFFFFh for 32-bit applications  0h to FFFFFFFFFFFFFFFFh for 64-bit applications  <b>VI_ATTR_RET_COUNT_32:</b> 0h to FFFFFFFFh  <b>VI_ATTR_RET_COUNT_64:</b> 0h to FFFFFFFFFFFFFFFFh	N/A

## Description

VI\_ATTR\_RET\_COUNT, VI\_ATTR\_RET\_COUNT\_32, and VI\_ATTR\_RET\_COUNT\_64 contain the actual number of elements that were asynchronously transferred.

VI\_ATTR\_RET\_COUNT\_32 is always a 32-bit value.

VI\_ATTR\_RET\_COUNT\_64 is always a 64-bit value. VI\_ATTR\_RET\_COUNT\_64 is not supported with 32-bit applications.

VI\_ATTR\_RET\_COUNT is a 32-bit value for 32-bit applications and a 64-bit value for 64-bit applications.

---

**Related Topics**

[VI\\_ATTR\\_BUFFER](#)

[VI\\_ATTR\\_JOB\\_ID](#)

[VI\\_ATTR\\_STATUS](#)

[VI\\_EVENT\\_IO\\_COMPLETION](#)

# VI\_ATTR\_RM\_SESSION

## Resource Classes

All I/O session types

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Local	ViSession	N/A	N/A

## Description

VI\_ATTR\_RM\_SESSION specifies the session of the Resource Manager that was used to open this session.

---

**Related Topics**

[VISA Resource Template](#)

## VI\_ATTR\_RSRC\_CLASS

### Resource Classes

All I/O session types

### Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViString	N/A	N/A

### Description

`VI_ATTR_RSRC_CLASS` specifies the resource class (for example, "INSTR") as defined by the canonical resource name.

---

### Related Topics

[VI\\_ATTR\\_RSRC\\_NAME](#)

[VISA Resource Template](#)

## VI\_ATTR\_RSRC\_IMPL\_VERSION

### Resource Classes

All I/O session types, all event object types, VISA Resource Manager

### Attribute Information

Access Privilege	Data Type	Range	Default
------------------	-----------	-------	---------

Read Only Global	ViVersion	0h to FFFFFFFFh	N/A
---------------------	-----------	-----------------	-----

Description

VI\_ATTR\_RSRC\_IMPL\_VERSION is the resource version that uniquely identifies each of the different revisions or implementations of a resource. This attribute value is defined by the individual manufacturer and increments with each new revision. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version.

Related Topics

[VI\\_ATTR\\_RSRC\\_SPEC\\_VERSION](#)

[VISA Resource Template](#)

VI\_ATTR\_RSRC\_LOCK\_STATE

Resource Classes

All I/O session types

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViAccessMode	VI_NO_LOCK (0) VI_EXCLUSIVE_LOCK (1) VI_SHARED_LOCK (2)	VI_NO_LOCK

# Description

VI\_ATTR\_RSRC\_LOCK\_STATE indicates the current locking state of the resource. The resource can be unlocked, locked with an exclusive lock, or locked with a shared lock.

---

## Related Topics

[VISA Resource Template](#)

# VI\_ATTR\_RSRC\_MANF\_ID

## Resource Classes

All I/O session types, all event object types, VISA Resource Manager

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0h to 3FFFh	N/A

# Description

VI\_ATTR\_RSRC\_MANF\_ID is a value that corresponds to the VXI manufacturer ID of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.

---

## Related Topics

[VI\\_ATTR\\_RSRC\\_MANF\\_NAME](#)

[VISA Resource Template](#)

# VI\_ATTR\_RSRC\_MANF\_NAME

## Resource Classes

All I/O session types, all event object types, VISA Resource Manager

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViString	N/A	N/A

## Description

VI\_ATTR\_RSRC\_MANF\_NAME is a string that corresponds to the manufacturer name of the vendor that implemented the VISA library. This attribute is not related to the device manufacturer attributes.



**Note** The value of this attribute is for display purposes only and not for programmatic decisions, as the value can differ between VISA implementations and/or revisions.

---

## Related Topics

[VI\\_ATTR\\_RSRC\\_MANF\\_ID](#)

[VISA Resource Template](#)

# VI\_ATTR\_RSRC\_NAME

# Resource Classes

All I/O session types

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViRsrc	N/A	N/A

## Description

VI\_ATTR\_RSRC\_NAME is the unique identifier for a resource. Refer to [VISA Resource Syntax and Examples](#) for the syntax of resource strings and examples.

---

### Related Topics

[viFindRsrc](#)

[viOpen](#)

[viParseRsrc](#)

[VISA Resource Template](#)

# VI\_ATTR\_RSRC\_SPEC\_VERSION

## Resource Classes

All I/O session types, all event object types, VISA Resource Manager

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViVersion	0h to FFFFFFFFh	00300000h

## Description

`VI_ATTR_RSRC_SPEC_VERSION` is the resource version that uniquely identifies the version of the VISA specification to which the implementation is compliant. The format of the value has the upper 12 bits as the major number of the version, the next lower 12 bits as the minor number of the version, and the lowest 8 bits as the sub-minor number of the version. The current VISA specification defines the value to be 00300000h.

## Related Topics

[VI\\_ATTR\\_RSRC\\_IMPL\\_VERSION](#)

[VISA Resource Template](#)

## VI\_ATTR\_SEND\_END\_EN

## Resource Classes

GPIO INSTR, GPIO INTFC, Serial INSTR, TCPIP INSTR, USB INSTR, VXI INSTR, VXI SERVANT

## Attribute Information

Access Privilege	Data Type	Range	Default
------------------	-----------	-------	---------



Read/Write Local	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_TRUE
---------------------	-----------	-----------------------------	---------

## Description

VI\_ATTR\_SEND\_END\_EN specifies whether to assert END during the transfer of the last byte of the buffer.

VI\_ATTR\_SEND\_END\_EN is relevant only in `viWrite` and related operations.

On Serial INSTR sessions, if this attribute is set to VI\_FALSE, the write will transmit the exact contents of the user buffer, without modifying it and without appending anything to the data being written. If this attribute is set to VI\_TRUE, VISA will perform the behavior described in VI\_ATTR\_ASRL\_END\_OUT.

On GPIB, VXI, TCP/IP INSTR, and USB INSTR sessions, if this attribute is set to VI\_TRUE, VISA will include the 488.2 defined "end of message" terminator.

---

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[VI\\_ATTR\\_ASRL\\_END\\_OUT](#)

[viWrite](#)

VI\_ATTR\_SIGP\_STATUS\_ID

## Resource Classes

VI\_EVENT\_VXI\_SIGP

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViUInt16	0h to FFFFh	N/A

## Description

VI\_ATTR\_SIGP\_STATUS\_ID is the 16-bit Status/ID value retrieved during the IACK cycle or from the Signal register.

---

## Related Topics

[INSTR Resource](#)

[VI\\_EVENT\\_VXI\\_SIGP](#)

## VI\_ATTR\_SLOT

## Resource Classes

PXI INSTR, VXI INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VXI	N/A

		<div>0 to 12</div> <div>VI_UNKNOWN_SLOT (-1)</div> <div>PXI</div> <div>1 to 18</div> <div>VI_UNKNOWN_SLOT (-1)</div>	
--	--	--	--

Description

VI\_ATTR\_SLOT specifies the physical slot location of the device. If the slot number is not known, VI\_UNKNOWN\_SLOT is returned.

Related Topics

[INSTR Resource](#)

VI\_ATTR\_SRC\_ACCESS\_PRIV

Resource Classes

VXI INSTR, VXI MEMACC


Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	<div>VI_DATA_PRIV (0)</div> <div>VI_DATA_NPRIV (1)</div> <div>VI_PROG_PRIV (2)</div> <div>VI_PROG_NPRIV (3)</div> <div>VI_BLK_PRIV (4)</div> <div>VI_BLK_NPRIV (5)</div> <div>VI_D64_PRIV (6)</div>	VI_DATA_PRIV

		VI_D64_NPRIV (7)	
--	--	------------------	--

Description

VI\_ATTR\_SRC\_ACCESS\_PRIV specifies the address modifier to be used in high-level access operations, such as viInXX() and viMoveInXX(), when reading from the source.



**Note** The values VI\_D64\_PRIV (6) and VI\_D64\_NPRIV (7) apply to only the block move operations. If you set this attribute to one of these values and then call one of the viInXX() operations, the operation returns VI\_ERROR\_INV\_SETUP.

Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[VI\\_ATTR\\_DEST\\_ACCESS\\_PRIV](#)

[VI\\_ATTR\\_SRC\\_BYTE\\_ORDER](#)

[VI\\_ATTR\\_SRC\\_INCREMENT](#)

[VI\\_ATTR\\_WIN\\_ACCESS\\_PRIV](#)

VI\_ATTR\_SRC\_BYTE\_ORDER

Resource Classes

VXI INSTR, VXI MEMACC

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_BIG_ENDIAN (0) VI_LITTLE_ENDIAN (1)	VI_BIG_ENDIAN

## Description

`VI_ATTR_SRC_BYTE_ORDER` specifies the byte order to be used in high-level access operations, such as `viInXX()` and `viMoveInXX()`, when reading from the source.

---

## Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[VI\\_ATTR\\_DEST\\_BYTE\\_ORDER](#)

[VI\\_ATTR\\_SRC\\_ACCESS\\_PRIV](#)

[VI\\_ATTR\\_SRC\\_INCREMENT](#)

[VI\\_ATTR\\_WIN\\_BYTE\\_ORDER](#)

## VI\_ATTR\_SRC\_INCREMENT

## Resource Classes

PXI INSTR, PXI MEMACC, VXI INSTR, VXI MEMACC

# Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViInt32	0 to 1	1

## Description

VI\_ATTR\_SRC\_INCREMENT is used in the viMoveInXX() operations to specify by how many elements the source offset is to be incremented after every transfer. The default value of this attribute is 1 (that is, the source address will be incremented by 1 after each transfer), and the viMoveInXX() operations move from consecutive elements. If this attribute is set to 0, the viMoveInXX() operations will always read from the same element, essentially treating the source as a FIFO register.

---

### Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[VI\\_ATTR\\_DEST\\_INCREMENT](#)

[VI\\_ATTR\\_SRC\\_ACCESS\\_PRIV](#)

[VI\\_ATTR\\_SRC\\_BYTE\\_ORDER](#)

## VI\_ATTR\_STATUS

### Resource Classes

VI\_EVENT\_EXCEPTION, VI\_EVENT\_IO\_COMPLETION, VI\_EVENT\_USB\_INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViStatus	N/A	N/A

## Description

`VI_ATTR_STATUS` contains the return code of the operation generating this event.

---

## Related Topics

[VI\\_ATTR\\_BUFFER](#)

[VI\\_ATTR\\_JOB\\_ID](#)

[VI\\_ATTR\\_MODEL\\_NAME](#)

[VI\\_ATTR\\_OPER\\_NAME](#)

[VI\\_ATTR\\_RET\\_COUNT/VI\\_ATTR\\_RET\\_COUNT\\_32/VI\\_ATTR\\_RET\\_COUNT\\_64](#)

[VI\\_EVENT\\_EXCEPTION](#)

[VI\\_EVENT\\_IO\\_COMPLETION](#)

[VI\\_EVENT\\_USB\\_INTR](#)

## `VI_ATTR_SUPPRESS_END_EN`

## Resource Classes

Serial INSTR, TCPIP SOCKET, USB RAW, VXI INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_FALSE

## Description

`VI_ATTR_SUPPRESS_END_EN` is relevant only in `viRead` and related operations.

For all session types on which this attribute is supported, if this attribute is set to `VI_TRUE`, read will not terminate due to an END condition. However, a read may still terminate successfully if `VI_ATTR_TERMCHAR_EN` is set to `VI_TRUE`. Otherwise, read will not terminate until all requested data is received (or an error occurs).

On Serial INSTR sessions, if this attribute is set to `VI_FALSE`, VISA will perform the behavior described in `VI_ATTR_ASRL_END_IN`.

On USB RAW sessions, if this attribute is set to `VI_FALSE`, VISA will perform the behavior described in `VI_ATTR_USB_END_IN`.

On TCP/IP SOCKET sessions, if this attribute is set to `VI_FALSE`, if NI-VISA reads some data and then detects a pause in the arrival of data packets, it will terminate the read operation. On TCP/IP SOCKET sessions, this attribute defaults to `VI_TRUE` in NI-VISA.

On VXI INSTR sessions, if this attribute is set to `VI_FALSE`, the END bit terminates read operations.

---

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_END\\_IN](#)

[VI\\_ATTR\\_USB\\_END\\_IN](#)



[viRead](#)

# VI\_ATTR\_TCPIP\_ADDR

## Resource Classes

TCPIP INSTR, TCPIP SOCKET

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViString	N/A	N/A

## Description

This is the TCPIP address of the device to which the session is connected. This string is formatted in dot notation.

---

## Related Topics

[INSTR Resource](#)

[SOCKET Resource](#)

[VI\\_ATTR\\_TCPIP\\_HOSTNAME](#)

# VI\_ATTR\_TCPIP\_DEVICE\_NAME

## Resource Classes

TCPIP INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViString	N/A	N/A

## Description

This specifies the LAN device name used by the VXI-11 or LXI protocol during connection.

---

## Related Topics

[INSTR Resource](#)

## VI\_ATTR\_TCPIP\_HOSTNAME

## Resource Classes

TCPIP INSTR, TCPIP SOCKET

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViString	N/A	N/A

## Description

This specifies the host name of the device. If no host name is available, this attribute returns an empty string.

Related Topics

[INSTR Resource](#)

[SOCKET Resource](#)

[VI\\_ATTR\\_TCPIP\\_ADDR](#)

VI\_ATTR\_TCPIP\_KEEPALIVE

Resource Classes

TCPIP SOCKET

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_FALSE

Description

Setting this attribute to TRUE requests that a TCP/IP provider enable the use of keep-alive packets on TCP connections. After the system detects that a connection was dropped, VISA returns a lost connection error code on subsequent I/O calls on the session. The time required for the system to detect that the connection was dropped is dependent on the system and is not settable.

Related Topics

[SOCKET Resource](#)

[VI\\_ATTR\\_TCPIP\\_NODELAY](#)

# VI\_ATTR\_TCPIP\_NODELAY

## Resource Classes

TCPIP SOCKET

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_TRUE

## Description

The Nagle algorithm is disabled when this attribute is enabled (and vice versa). The Nagle algorithm improves network performance by buffering "send" data until a full-size packet can be sent. This attribute is enabled by default in VISA to verify that synchronous writes get flushed immediately.

---

## Related Topics

[SOCKET Resource](#)

[VI\\_ATTR\\_TCPIP\\_KEEPA](#)  
[LIVE](#)

# VI\_ATTR\_TCPIP\_PORT

## Resource Classes

TCPIP SOCKET

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	0 to FFFFh	N/A

## Description

This specifies the port number for a given TCPIP address. For a TCPIP SOCKET Resource, this is a required part of the address string.

## Related Topics

[SOCKET Resource](#)

## VI\_ATTR\_TERMCHAR

## Resource Classes

GPIO INSTR, GPIO INTRFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt8	0 to FFh	0Ah (linefeed)

## Description

VI\_ATTR\_TERMCHAR is the termination character. When the termination character is

read and `VI_ATTR_TERMCHAR_EN` is enabled during a read operation, the read operation terminates.

For a Serial INSTR session, `VI_ATTR_TERMCHAR` is Read/Write when the corresponding session is not enabled to receive `VI_EVENT_ASRL_TERMCHAR` events. When the session is enabled to receive `VI_EVENT_ASRL_TERMCHAR` events, the attribute `VI_ATTR_TERMCHAR` is Read Only. For all other session types, the attribute `VI_ATTR_TERMCHAR` is always Read/Write.

---

**Related Topics**

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[VI\\_ATTR\\_TERMCHAR\\_EN](#)

# VI\_ATTR\_TERMCHAR\_EN

## Resource Classes

GPIO INSTR, GPIO INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViBoolean	VI_TRUE (1) VI_FALSE (0)	VI_FALSE

# Description

VI\_ATTR\_TERMCHAR\_EN is a flag that determines whether the read operation should terminate when a termination character is received. This attribute is ignored if VI\_ATTR\_ASRL\_END\_IN is set to VI\_ASRL\_END\_TERMCHAR. This attribute is valid for both raw I/O (viRead) and formatted I/O (viScanf).

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[VI\\_ATTR\\_TERMCHAR](#)

# VI\_ATTR\_TMO\_VALUE

## Resource Classes

All I/O session types

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt32	VI_TMO_IMMEDIATE (0); 1 to FFFFFFFEh; VI_TMO_INFINITE (FFFFFFFFh)	2000

## Description

`VI_ATTR_TMO_VALUE` specifies the minimum timeout value to use (in milliseconds) when accessing the device associated with the given session. A timeout value of `VI_TMO_IMMEDIATE` means that operations should never wait for the device to respond. A timeout value of `VI_TMO_INFINITE` disables the timeout mechanism.

Notice that the actual timeout value used by the driver may be higher than the requested one. The actual timeout value is returned when this attribute is retrieved via `viGetAttribute()`.

---

## Related Topics

[BACKPLANE Resource](#)

[INSTR Resource](#)

[INTFC Resource](#)

[MEMACC Resource](#)

[Resources](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

## VI\_ATTR\_TRIG\_ID

## Resource Classes

GPIB INSTR, GPIB INTFC, PXI INSTR, PXI BACKPLANE, Serial INSTR, TCPIP INSTR, VXI BACKPLANE, VXI INSTR, VXI SERVANT



## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViInt16	<b>GPIO, Serial, TCPIP:</b> VI_TRIG_SW (-1)	VI_TRIG_SW
		<b>VXI:</b> VI_TRIG_SW (-1); VI_TRIG_TTL0 (0) to V I_TRIG_TTL7 (7); VI_TRIG_ECL0 (8) to V I_TRIG_ECL1 (9)	VI_TRIG_SW
		<b>PXI INSTR, PXI BACKPLANE:</b> VI_TRIG_SW (-1) VI_TRIG_TTL0 (0) to V I_TRIG_TTL7 (7)	VI_TRIG_SW

## Description

VI\_ATTR\_TRIG\_ID is the identifier for the current triggering mechanism.

VI\_ATTR\_TRIG\_ID is Read/Write when the corresponding session is not enabled to receive trigger events. When the session is enabled to receive trigger events, the attribute VI\_ATTR\_TRIG\_ID is Read Only.

## Related Topics

[BACKPLANE Resource](#)

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[VI\\_ATTR\\_RECV\\_TRIG\\_ID](#)

[viAssertTrigger](#)

VI\_ATTR\_USB\_ALT\_SETTING

Resource Classes

USB RAW

Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViInt16	0 to FFh	0

Description

VI\_ATTR\_USB\_ALT\_SETTING specifies the USB alternate setting used by this USB interface.

VI\_ATTR\_USB\_ALT\_SETTING is Read/Write when the corresponding session is not enabled to receive USB interrupt events. If the session is enabled to receive USB interrupt events or if there are any other sessions to this resource, the attribute VI\_ATTR\_USB\_ALT\_SETTING is Read Only.

---

Related Topics

[RAW Resource](#)

VI\_ATTR\_USB\_BULK\_IN\_PIPE

# Resource Classes

USB RAW

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViInt16	-1, 81h to 8Fh	N/A

## Description

`VI_ATTR_USB_BULK_IN_PIPE` specifies the endpoint address of the USB bulk-in pipe used by the given session. An initial value of -1 signifies that this resource does not have any bulk-in pipes. This endpoint is used in `viRead` and related operations.

---

### Related Topics

[RAW Resource](#)

[VI\\_ATTR\\_USB\\_BULK\\_OUT\\_PIPE](#)

[VI\\_ATTR\\_USB\\_CTRL\\_PIPE](#)

[VI\\_ATTR\\_USB\\_INTR\\_IN\\_PIPE](#)

[VI\\_ATTR\\_USB\\_NUM\\_PIPES](#)

## VI\_ATTR\_USB\_BULK\_IN\_STATUS

# Resource Classes

USB RAW

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViInt16	VI_USB_PIPE_STATE_UNKNOWN (-1) VI_USB_PIPE_READY (0) VI_USB_PIPE_STALLED (1)	N/A

## Description

VI\_ATTR\_USB\_BULK\_IN\_STATUS specifies whether the USB bulk-in pipe used by the given session is stalled or ready. This attribute can be set to only VI\_USB\_PIPE\_READY.

---

## Related Topics

[RAW Resource](#)

# VI\_ATTR\_USB\_BULK\_OUT\_PIPE

## Resource Classes

USB RAW

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViInt16	-1, 01h to 0Fh	N/A

## Description

`VI_ATTR_USB_BULK_OUT_PIPE` specifies the endpoint address of the USB bulk-out or interrupt-out pipe used by the given session. An initial value of `-1` signifies that this resource does not have any bulk-out or interrupt-out pipes. This endpoint is used in `viWrite` and related operations.

---

### Related Topics

[RAW Resource](#)

[VI\\_ATTR\\_USB\\_BULK\\_IN\\_PIPE](#)

[VI\\_ATTR\\_USB\\_CTRL\\_PIPE](#)

[VI\\_ATTR\\_USB\\_INTR\\_IN\\_PIPE](#)

[VI\\_ATTR\\_USB\\_NUM\\_PIPES](#)

## VI\_ATTR\_USB\_BULK\_OUT\_STATUS

### Resource Classes

USB RAW

### Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViInt16	VI_USB_PIPE_STATE_UNKNOWN (-1) VI_USB_PIPE_READY (0) VI_USB_PIPE_STALLED (1)	N/A

## Description

`VI_ATTR_USB_BULK_OUT_STATUS` specifies whether the USB bulk-out or interrupt-out pipe used by the given session is stalled or ready. This attribute can be set to only `VI_USB_PIPE_READY`.

---

### Related Topics

[RAW Resource](#)

## VI\_ATTR\_USB\_CLASS

### Resource Classes

USB RAW

### Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	0 to FFh	N/A

## Description

`VI_ATTR_USB_CLASS` specifies the USB class used by this USB interface.

---

### Related Topics

[RAW Resource](#)

## VI\_ATTR\_USB\_CTRL\_PIPE

## Resource Classes

### USB RAW

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViInt16	00h to 0Fh	00h

## Description

`VI_ATTR_USB_CTRL_PIPE` specifies the endpoint address of the USB control pipe used by the given session. A value of 0 signifies that the default control pipe will be used. This endpoint is used in `viUsbControlIn` and `viUsbControlOut` operations. Nonzero values may not be supported on all platforms.

---

## Related Topics

[RAW Resource](#)

[VI\\_ATTR\\_USB\\_BULK\\_IN\\_PIPE](#)

[VI\\_ATTR\\_USB\\_BULK\\_OUT\\_PIPE](#)

[VI\\_ATTR\\_USB\\_INTR\\_IN\\_PIPE](#)

[VI\\_ATTR\\_USB\\_NUM\\_PIPES](#)

[viUsbControlIn](#)

[viUsbControlOut](#)

`VI_ATTR_USB_END_IN`

# Resource Classes

## USB RAW

### Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_USB_END_NONE (0) VI_USB_END_SHORT (4) VI_USB_END_SHORT_OR_COUNT (5)	VI_USB_END_SHORT_OR_COUNT

### Description

VI\_ATTR\_USB\_END\_IN indicates the method used to terminate read operations.

If it is set to VI\_USB\_END\_NONE, short packets are ignored for read operations, so reads will not terminate until all of the requested data is received (or an error occurs).

If it is set to VI\_USB\_END\_SHORT, the read operation will terminate on a short packet; use this if the device will terminate all read transfers with a short packet, including sending a zero (short) packet when the last data packet is full.

If it is set to VI\_USB\_END\_SHORT\_OR\_COUNT, the read operation will terminate on a short packet or when it receives the requested count of data bytes; use this if the device does not send zero packets.

---

### Related Topics

[RAW Resource](#)

## VI\_ATTR\_USB\_INTFC\_NUM



## Resource Classes

USB INSTR, USB RAW

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	0 to FEh	0

## Description

`VI_ATTR_USB_INTFC_NUM` specifies the USB interface number used by the given session.

---

## Related Topics

[INSTR Resource](#)

[RAW Resource](#)

## VI\_ATTR\_USB\_INTR\_IN\_PIPE

## Resource Classes

USB RAW

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViInt16	-1, 81h to 8Fh	N/A

## Description

`VI_ATTR_USB_INTR_IN_PIPE` specifies the endpoint address of the USB interrupt-in pipe used by the given session. An initial value of -1 signifies that this resource does not have any interrupt-in pipes. This endpoint is used in `viEnableEvent` for `VI_EVENT_USB_INTR`.

---

## Related Topics

[RAW Resource](#)

[VI\\_ATTR\\_USB\\_BULK\\_IN\\_PIPE](#)

[VI\\_ATTR\\_USB\\_BULK\\_OUT\\_PIPE](#)

[VI\\_ATTR\\_USB\\_CTRL\\_PIPE](#)

[VI\\_ATTR\\_USB\\_NUM\\_PIPES](#)

## VI\_ATTR\_USB\_INTR\_IN\_STATUS

## Resource Classes

USB RAW

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViInt16	VI_USB_PIPE_STATE_UNKNOWN (-1) VI_USB_PIPE_READY (0) VI_USB_PIPE_STALLED (1)	N/A

# Description

VI\_ATTR\_USB\_INTR\_IN\_STATUS specifies whether the USB interrupt-in pipe used by the given session is stalled or ready. This attribute can be set to only VI\_USB\_PIPE\_READY.

---

## Related Topics

[RAW Resource](#)

# VI\_ATTR\_USB\_MAX\_INTR\_SIZE

## Resource Classes

USB INSTR, USB RAW

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	0 to FFFFh	N/A

# Description

VI\_ATTR\_USB\_MAX\_INTR\_SIZE specifies the maximum size of data that will be stored by any given USB interrupt. If a USB interrupt contains more data than this size, the data in excess of this size will be lost.

VI\_ATTR\_USB\_MAX\_INTR\_SIZE is Read/Write when the corresponding session is not enabled to receive USB interrupt events. When the session is enabled to receive USB interrupt events, the attribute VI\_ATTR\_USB\_MAX\_INTR\_SIZE is Read Only.

---

## Related Topics

[INSTR Resource](#)

[RAW Resource](#)

[VI\\_EVENT\\_USB\\_INTR](#)

# VI\_ATTR\_USB\_NUM\_INTFCS

Resource Classes

USB RAW

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	1 to FFh	N/A

Description

VI\_ATTR\_USB\_NUM\_INTFCS specifies the number of interfaces supported by this USB device.

---

Related Topics

[RAW Resource](#)

# VI\_ATTR\_USB\_NUM\_PIPES

Resource Classes

USB RAW

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	0 to 30	N/A

## Description

`VI_ATTR_USB_NUM_PIPES` specifies the number of pipes supported by this USB interface. This does not include the default control pipe.

---

## Related Topics

[RAW Resource](#)

[VI\\_ATTR\\_USB\\_BULK\\_IN\\_PIPE](#)

[VI\\_ATTR\\_USB\\_BULK\\_OUT\\_PIPE](#)

[VI\\_ATTR\\_USB\\_INTR\\_IN\\_PIPE](#)

## VI\_ATTR\_USB\_PROTOCOL

## Resource Classes

USB INSTR, USB RAW

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	0 to FFh	N/A

## Description

`VI_ATTR_USB_PROTOCOL` specifies the USB protocol used by this USB interface.

---

### Related Topics

[INSTR Resource](#)

[RAW Resource](#)

## VI\_ATTR\_USB\_RECV\_INTR\_DATA

### Resource Classes

`VI_EVENT_USB_INTR`

### Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViAUInt8	N/A	N/A

## Description

`VI_ATTR_USB_RECV_INTR_DATA` contains the actual received data from the USB Interrupt. The passed in data buffer **must** be of size at least equal to the value of `VI_ATTR_USB_RECV_INTR_SIZE`.

---

### Related Topics

[VI\\_ATTR\\_USB\\_RECV\\_INTR\\_SIZE](#)

[VI\\_EVENT\\_USB\\_INTR](#)

# VI\_ATTR\_USB\_RECV\_INTR\_SIZE

## Resource Classes

VI\_EVENT\_USB\_INTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only	ViUInt16	N/A	N/A

## Description

VI\_ATTR\_USB\_RECV\_INTR\_SIZE contains the number of bytes of USB interrupt data that is stored.

---

## Related Topics

[VI\\_ATTR\\_USB\\_RECV\\_INTR\\_DATA](#)

[VI\\_EVENT\\_USB\\_INTR](#)

# VI\_ATTR\_USB\_SERIAL\_NUM

## Resource Classes

USB INSTR, USB RAW

## Attribute Information

Access Privilege	Data Type	Range	Default
------------------	-----------	-------	---------

Read Only Global	ViString	N/A	N/A
------------------	----------	-----	-----

Description

VI\_ATTR\_USB\_SERIAL\_NUM specifies the USB serial number of this device.

---

Related Topics

[INSTR Resource](#)

[RAW Resource](#)

VI\_ATTR\_USB\_SUBCLASS

Resource Classes

USB RAW

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	0 to FFh	N/A

Description

VI\_ATTR\_USB\_SUBCLASS specifies the USB subclass used by this USB interface.

---

Related Topics

[RAW Resource](#)



# VI\_ATTR\_USER\_DATA/VI\_ATTR\_USER\_DATA\_32/ VI\_ATTR\_USER\_DATA\_64

## Resource Classes

All I/O session types

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/ Write Local	<b>VI_ATTR_USER_DATA:</b> ViAddr	<b>VI_ATTR_USER_DATA:</b> Not specified	N/A
	<b>VI_ATTR_USER_DATA_32:</b> ViUInt32	<b>VI_ATTR_USER_DATA_32:</b> 0h to FFFFFFFFh	
	<b>VI_ATTR_USER_DATA_64:</b> ViUInt64	<b>VI_ATTR_USER_DATA_64:</b> 0h to FFFFFFFFFFFFFFFFh	

## Description

VI\_ATTR\_USER\_DATA, VI\_ATTR\_USER\_DATA\_32, and VI\_ATTR\_USER\_DATA\_64 store data to be used privately by the application for a particular session. VISA does not use this data for any purpose. It is provided to the application for its own use.

VI\_ATTR\_USER\_DATA\_64 is not supported with 32-bit applications.

---

## Related Topics

[VISA Resource Template](#)

## VI\_ATTR\_VXI\_DEV\_CLASS

# Resource Classes

VXI INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt16	VI_VXI_CLASS_MEMORY (0) VI_VXI_CLASS_EXTENDED (1) VI_VXI_CLASS_MESSAGE (2) VI_VXI_CLASS_REGISTER (3) VI_VXI_CLASS_OTHER (4)	N/A

## Description

This attribute represents the VXI-defined device class to which the resource belongs, either message based (VI\_VXI\_CLASS\_MESSAGE), register based (VI\_VXI\_CLASS\_REGISTER), extended (VI\_VXI\_CLASS\_EXTENDED), or memory (VI\_VXI\_CLASS\_MEMORY). VME devices are usually either register based or belong to a miscellaneous class (VI\_VXI\_CLASS\_OTHER).

## Related Topics

[INSTR Resource](#)

VI\_ATTR\_VXI\_LA

# Resource Classes

VXI INSTR, VXI MEMACC, VXI SERVANT

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	0 to 511	N/A

## Description

For an INSTR session, `VI_ATTR_VXI_LA` specifies the logical address of the VXI or VME device used by the given session. For a MEMACC or SERVANT session, this attribute specifies the logical address of the local controller.

---

## Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[SERVANT Resource](#)

## VI\_ATTR\_TRIG\_DIR

## Resource Classes

VXI INSTR

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViUInt16	N/A	0

## Description

`VI_ATTR_TRIG_DIR` is a bit map of the directions of the mapped TTL trigger lines. Bits 0-7 represent TTL triggers 0-7 respectively. A bit's value of 0 means the line is routed out of the frame, and a value of 1 means into the frame. In order for a direction to be set, the line must also be enabled using `VI_ATTR_VXI_TRIG_LINES_EN`.

---

### Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_VXI\\_TRIG\\_LINES\\_EN](#)

## VI\_ATTR\_VXI\_TRIG\_LINES\_EN

### Resource Classes

VXI INSTR

### Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Global	ViUInt16	N/A	0

## Description

`VI_ATTR_VXI_TRIG_LINES_EN` is a bit map of what VXI TLL triggers have mappings. Bits 0-7 represent TTL triggers 0-7 respectively. A bit's value of 0 means the trigger line is unmapped, and 1 means a mapping exists. Use `VI_ATTR_VXI_TRIG_DIR` to set an enabled line's direction.

---

Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_VXI\\_TRIG\\_DIR](#)

VI\_ATTR\_VXI\_TRIG\_STATUS

Resource Classes

VXI BACKPLANE

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt32	N/A	N/A

Description

This attribute shows the current state of the VXI trigger lines. This is a bit vector with bits 0-9 corresponding to `VI_TRIG_TTL0` through `VI_TRIG_ECL1`.

---

Related Topics

[BACKPLANE Resource](#)

[VI\\_ATTR\\_VXI\\_TRIG\\_SUPPORT](#)

[VI\\_ATTR\\_VXI\\_VME\\_INTR\\_STATUS](#)

[VI\\_ATTR\\_VXI\\_VME\\_SYSFAIL\\_STATE](#)

# VI\_ATTR\_VXI\_TRIG\_SUPPORT

## Resource Classes

VXI INSTR, VXI BACKPLANE

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViUInt32	N/A	N/A

## Description

This attribute shows which VXI trigger lines this implementation supports. This is a bit vector with bits 0-9 corresponding to `VI_TRIG_TTL0` through `VI_TRIG_ECL1`.

---

## Related Topics

[BACKPLANE Resource](#)

[INSTR Resource](#)

# VI\_ATTR\_VXI\_VME\_INTR\_STATUS

## Resource Classes

VXI BACKPLANE

## Attribute Information

Access Privilege	Data Type	Range	Default
------------------	-----------	-------	---------

Read Only Global	ViUInt16	N/A	N/A
---------------------	----------	-----	-----

Description

This attribute shows the current state of the VXI/VME interrupt lines. This is a bit vector with bits 0-6 corresponding to interrupt lines 1-7.

Related Topics

[BACKPLANE Resource](#)

[VI\\_ATTR\\_VXI\\_TRIG\\_STATUS](#)

[VI\\_ATTR\\_VXI\\_VME\\_SYSFAIL\\_STATE](#)

VI\_ATTR\_VXI\_VME\_SYSFAIL\_STATE

Resource Classes

VXI BACKPLANE

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Global	ViInt16	VI_STATE_ASSERTED(1) VI_STATE_DEASSERTED(0) VI_STATE_UNKNOWN(-1)	N/A

## Description

This attribute shows the current state of the VXI/VME SYSFAIL (SYStem FAILure) backplane line.

---

### Related Topics

[BACKPLANE Resource](#)

[VI\\_ATTR\\_VXI\\_TRIG\\_STATUS](#)

[VI\\_ATTR\\_VXI\\_VME\\_INTR\\_STATUS](#)

## VI\_ATTR\_WIN\_ACCESS

### Resource Classes

PXI INSTR, PXI MEMACC, VXI INSTR, VXI MEMACC

### Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Local	ViUInt16	VI_NMAPPED (1) VI_USE_OPERS (2) VI_DEREF_ADDR (3)	VI_NMAPPED

## Description

VI\_ATTR\_WIN\_ACCESS specifies the modes in which the current window may be accessed.

- If VI\_NMAPPED, the window is not currently mapped.
- If VI\_USE\_OPERS, the window is accessible through the viPeekXX() and viPo



keXX() operations only.

- If VI\_DEREF\_ADDR, you can either use operations or directly dereference the mapped address as a pointer.

---

**Related Topics**

[INSTR Resource](#)

[MEMACC Resource](#)

[VI\\_ATTR\\_WIN\\_ACCESS\\_PRIV](#)

[VI\\_ATTR\\_WIN\\_BASE\\_ADDR/VI\\_ATTR\\_WIN\\_BASE\\_ADDR\\_32/  
VI\\_ATTR\\_WIN\\_BASE\\_ADDR\\_64](#)

[VI\\_ATTR\\_WIN\\_BYTE\\_ORDER](#)

[VI\\_ATTR\\_WIN\\_SIZE/VI\\_ATTR\\_WIN\\_SIZE\\_32/VI\\_ATTR\\_WIN\\_SIZE\\_64](#)

[viMapAddress/viMapAddressEx](#)

[viPeek8/viPeek16/viPeek32/viPeek64](#)

[viPoke8/viPoke16/viPoke32/viPoke64](#)

**VI\_ATTR\_WIN\_ACCESS\_PRIV**

**Resource Classes**

VXI INSTR, VXI MEMACC

**Attribute Information**

Access Privilege	Data Type	Range	Default
------------------	-----------	-------	---------

Read/Write Local	ViUInt16	VI_DATA_PRIV (0) VI_DATA_NPRIV (1) VI_PROG_PRIV (2) VI_PROG_NPRIV (3) VI_BLK_PRIV (4) VI_BLK_NPRIV (5)	VI_DATA_PRIV
---------------------	----------	---	--------------

## Description

VI\_ATTR\_WIN\_ACCESS\_PRIV specifies the address modifier to be used in low-level access operations, such as `viMapAddress()`, `viPeekXX()`, and `viPokeXX()`, when accessing the mapped window.

This attribute is Read/Write when the corresponding session is not mapped (that is, when VI\_ATTR\_WIN\_ACCESS is VI\_NMAPPED. When the session is mapped, this attribute is Read Only.

---

### Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[VI\\_ATTR\\_DEST\\_ACCESS\\_PRIV](#)

[VI\\_ATTR\\_SRC\\_ACCESS\\_PRIV](#)

[VI\\_ATTR\\_WIN\\_ACCESS](#)

[VI\\_ATTR\\_WIN\\_BASE\\_ADDR/VI\\_ATTR\\_WIN\\_BASE\\_ADDR\\_32/  
VI\\_ATTR\\_WIN\\_BASE\\_ADDR\\_64](#)

[VI\\_ATTR\\_WIN\\_BYTE\\_ORDER](#)

[VI\\_ATTR\\_WIN\\_SIZE/VI\\_ATTR\\_WIN\\_SIZE\\_32/VI\\_ATTR\\_WIN\\_SIZE\\_64](#)

# VI\_ATTR\_WIN\_BASE\_ADDR/VI\_ATTR\_WIN\_BASE\_ADDR\_32/ VI\_ATTR\_WIN\_BASE\_ADDR\_64

## Resource Classes

PXI INSTR, PXI MEMACC, VXI INSTR, VXI MEMACC

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Local	<b>VI_ATTR_WIN_BASE_ADDR:</b> ViBusAddress  <b>VI_ATTR_WIN_BASE_ADDR_32:</b> ViUInt32  <b>VI_ATTR_WIN_BASE_ADDR_64:</b> ViUInt64	<b>VI_ATTR_WIN_BASE_ADDR:</b> 0h to FFFFFFFFh for 32-bit applications  0h to FFFFFFFFFFFFFFFFh for 64-bit applications  <b>VI_ATTR_WIN_BASE_ADDR_32:</b> 0h to FFFFFFFFh  <b>VI_ATTR_WIN_BASE_ADDR_64:</b> 0h to FFFFFFFFFFFFFFFFh	N/A

## Description

VI\_ATTR\_WIN\_BASE\_ADDR, VI\_ATTR\_WIN\_BASE\_ADDR\_32, and VI\_ATTR\_WIN\_BASE\_ADDR\_64 specify the base address of the interface bus to which this window is mapped. If the value of VI\_ATTR\_WIN\_ACCESS is VI\_NMAPPED, the value of this attribute is undefined.

## Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[VI\\_ATTR\\_WIN\\_ACCESS](#)

[VI\\_ATTR\\_WIN\\_ACCESS\\_PRIV](#)

[VI\\_ATTR\\_WIN\\_BYTE\\_ORDER](#)

[VI\\_ATTR\\_WIN\\_SIZE/VI\\_ATTR\\_WIN\\_SIZE\\_32/VI\\_ATTR\\_WIN\\_SIZE\\_64](#)

# VI\_ATTR\_WIN\_BYTE\_ORDER

## Resource Classes

VXI INSTR, VXI MEMACC

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_BIG_ENDIAN (0) VI_LITTLE_ENDIAN (1)	VI_BIG_ENDIAN

## Description

VI\_ATTR\_WIN\_BYTE\_ORDER specifies the byte order to be used in low-level access operations, such as viMapAddress(), viPeekXX(), and viPokeXX(), when accessing the mapped window.

This attribute is Read/Write when the corresponding session is not mapped (that is, when VI\_ATTR\_WIN\_ACCESS is VI\_NMAPPED. When the session is mapped, this attribute is Read Only.

Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[VI\\_ATTR\\_DEST\\_BYTE\\_ORDER](#)

[VI\\_ATTR\\_SRC\\_BYTE\\_ORDER](#)

[VI\\_ATTR\\_WIN\\_ACCESS](#)

[VI\\_ATTR\\_WIN\\_ACCESS\\_PRIV](#)

[VI\\_ATTR\\_WIN\\_BASE\\_ADDR/VI\\_ATTR\\_WIN\\_BASE\\_ADDR\\_32/  
VI\\_ATTR\\_WIN\\_BASE\\_ADDR\\_64](#)

[VI\\_ATTR\\_WIN\\_SIZE/VI\\_ATTR\\_WIN\\_SIZE\\_32/VI\\_ATTR\\_WIN\\_SIZE\\_64](#)

[VI\\_ATTR\\_WIN\\_SIZE/VI\\_ATTR\\_WIN\\_SIZE\\_32/  
VI\\_ATTR\\_WIN\\_SIZE\\_64](#)

Resource Classes

PXI INSTR, PXI MEMACC, VXI INSTR, VXI MEMACC

Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Local	<b>VI_ATTR_WIN_SIZE:</b> ViBusSize  <b>VI_ATTR_WIN_SIZE_32:</b>	<b>VI_ATTR_WIN_SIZE:</b> 0h to FFFFFFFFh for 32-bit applications  0h to FFFFFFFFFFFFFFFFh for 64-bit	N/A

		applications	
	ViUInt32		
	<b>VI_ATTR_WIN_SIZE_64:</b> ViUInt64	<b>VI_ATTR_WIN_SIZE_32:</b> 0h to FFFFFFFFh  <b>VI_ATTR_WIN_SIZE_64:</b> 0h to FFFFFFFFFFFFFFFFh	

Description

VI\_ATTR\_WIN\_SIZE, VI\_ATTR\_WIN\_SIZE\_32, and VI\_ATTR\_WIN\_SIZE\_64 specify the size of the region mapped to this window. If the value of VI\_ATTR\_WIN\_ACCESS is VI\_NMAPPED, the value of this attribute is undefined.

Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[VI\\_ATTR\\_WIN\\_ACCESS](#)

[VI\\_ATTR\\_WIN\\_ACCESS\\_PRIV](#)

[VI\\_ATTR\\_WIN\\_BASE\\_ADDR/VI\\_ATTR\\_WIN\\_BASE\\_ADDR\\_32/  
VI\\_ATTR\\_WIN\\_BASE\\_ADDR\\_64](#)

[VI\\_ATTR\\_WIN\\_BYTE\\_ORDER](#)

VI\_ATTR\_WR\_BUF\_OPER\_MODE

## Resource Classes

GPIO INSTR, GPIO INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

## Attribute Information

Access Privilege	Data Type	Range	Default
Read/Write Local	ViUInt16	VI_FLUSH_ON_ACCESS (1) VI_FLUSH_WHEN_FULL (2)	VI_FLUSH_WHEN_FULL

## Description

`VI_ATTR_WR_BUF_OPER_MODE` specifies the operational mode of the formatted I/O write buffer. When the operational mode is set to `VI_FLUSH_WHEN_FULL` (default), the buffer is flushed when an END indicator is written to the buffer, or when the buffer fills up. If the operational mode is set to `VI_FLUSH_ON_ACCESS`, the write buffer is flushed under the same conditions, and also every time a `viPrintf()` (or related) operation completes.

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[VI\\_ATTR\\_RD\\_BUF\\_OPER\\_MODE](#)

[viFlush](#)

[viPrintf](#)

# VI\_ATTR\_WR\_BUF\_SIZE

## Resource Classes

GPIB INSTR, GPIB INTRFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

## Attribute Information

Access Privilege	Data Type	Range	Default
Read Only Local	ViUInt32	N/A	N/A

## Description

This is the current size of the formatted I/O output buffer for this session. The user can modify this value by calling `viSetBuf()`.

---

## Related Topics

[VI\\_ATTR\\_RD\\_BUF\\_SIZE](#)

[VI\\_ATTR\\_WR\\_BUF\\_OPER\\_MODE](#)

[viSetBuf](#)




# Events

These topics describe the VISA events. The event descriptions are listed in alphabetical order for easy reference.

Each event description contains a list below the title indicating the supported resource classes, such as GPIB, Serial, etc. The event description contains a brief description of the event attributes. [Attributes](#) contains more detailed descriptions of the event attributes.

## VI\_EVENT\_ASRL\_BREAK



**Note** This event is supported for all serial ports on Windows and LabVIEW RT, and ENET-Serial on all platforms. Except for ENET-Serial, it is not supported for serial ports on Mac.

### Resource Classes

Serial INSTR

### Description

Notification that a break signal was received.

### Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.

---


### Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_BREAK\\_STATE](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

# VI\_EVENT\_ASRL\_CHAR



**Note** This event is supported for all serial ports on Windows and LabVIEW RT, and ENET-Serial on all platforms. Except for ENET-Serial, it is not supported for serial ports on Mac.

## Resource Classes

Serial INSTR

## Description

Notification that at least one data byte has been received. Each data character will not necessarily result in an event notification. In other words, if multiple data bytes arrive at once, you may get only one event. After receiving this event, you should query the serial port for the number of bytes available via the `VI_ATTR_ASRL_AVAIL_NUM` attribute.

## Event Attributes

Symbolic Name	Description
<code>VI_ATTR_EVENT_TYPE</code>	Unique logical identifier of the event.

---


## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_AVAIL\\_NUM](#)

VI\_ATTR\_EVENT\_TYPE

VI\_EVENT\_ASRL\_CTS



**Note** This event is supported for all serial ports on Windows and LabVIEW RT, and ENET-Serial on all platforms. Except for ENET-Serial, it is not supported for serial ports on Mac.

Resource Classes

Serial INSTR

Description

Notification that the Clear To Send (CTS) line changed state. If the CTS line changes state quickly several times in succession, not all line state changes will necessarily result in event notifications.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.

---

Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_CTS\\_STATE](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

VI\_EVENT\_ASRL\_DCD



**Note** This event is supported for all serial ports on Windows and LabVIEW RT, and ENET-Serial on all platforms. Except for ENET-Serial, it is not supported for serial ports on Mac.

# Resource Classes

Serial INSTR

## Description

Notification that the Data Carrier Detect (DCD) line changed state. If the DCD line changes state quickly several times in succession, not all line state changes will necessarily result in event notifications.

## Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.

---

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_DCD\\_STATE](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

## VI\_EVENT\_ASRL\_DSR



**Note** This event is supported for all serial ports on Windows and LabVIEW RT, and ENET-Serial on all platforms. Except for ENET-Serial, it is not supported for serial ports on Mac.

# Resource Classes

Serial INSTR

## Description

Notification that the Data Set Ready (DSR) line changed state. If the DSR line changes state quickly several times in succession, not all line state changes will necessarily result in event notifications.

## Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.

---

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_DSR\\_STATE](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

## VI\_EVENT\_ASRL\_RI



**Note** This event is supported for all serial ports on Windows and LabVIEW RT, and ENET-Serial on all platforms. Except for ENET-Serial, it is not supported for serial ports on Mac.

# Resource Classes

Serial INSTR

## Description

Notification that the Ring Indicator (RI) signal level was asserted.

## Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.

---

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_RI\\_STATE](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

## VI\_EVENT\_ASRL\_TERMCHAR



**Note** This event is supported for all serial ports on Windows and LabVIEW RT, and ENET-Serial on all platforms. Except for ENET-Serial, it is not supported for serial ports on Linux or Mac.

## Resource Classes

Serial INSTR

## Description

Notification that the termination character has been received. The actual termination character is specified by setting `VI_ATTR_TERMCHAR` prior to enabling this event. For this event, the setting of `VI_ATTR_TERMCHAR_EN` is ignored

## Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.

### Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_ASRL\\_AVAIL\\_NUM](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

[VI\\_ATTR\\_TERMCHAR](#)

## VI\_EVENT\_CLEAR

### Resource Classes

GPIB INTFC, VXI SERVANT

### Description

Notification that the local controller has been sent a device clear message.

### Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.

---

## Related Topics

[INTFC Resource](#)

[SERVANT Resource](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

## VI\_EVENT\_EXCEPTION

### Resource Classes

All I/O session types

### Description

This event notifies the application that an error condition has occurred during an operation invocation. In VISA, exceptions are defined as events. The exception-handling model follows the event-handling model for callbacks, and is like any other event in VISA, except that the queueing and suspended handler mechanisms are not allowed.

A VISA operation generating an exception blocks until the exception handler execution is completed. However, an exception handler sometimes may prefer to terminate the program prematurely without returning the control to the operation generating the exception. VISA does not preclude an application from using a platform-specific or language-specific exception handling mechanism from within the VISA exception handler. For example, the C++ try/catch block can be used in an application in conjunction with the C++ throw mechanism from within the VISA exception handler.

When using the C++ try/catch/throw or other exception-handling mechanisms, the control will not return to the VISA system. This has some important repercussions:

- If multiple handlers were installed on the exception event, the handlers that were not invoked prior to the current handler will not be invoked for the current exception.
- The exception context will not be deleted by the VISA system when a C++ exception



is used. In this case, the application should delete the exception context as soon as the application has no more use for the context, before terminating the session. An application should use the `viClose()` operation to delete the exception context.

One situation in which an exception event will not be generated is in the case of asynchronous operations. If the error is detected after the operation is posted—once the asynchronous portion has begun—the status is returned normally via the I/O completion event. However, if an error occurs before the asynchronous portion begins—the error is returned from the asynchronous operation itself—then the exception event will still be raised. This deviation is due to the fact that asynchronous operations already raise an event when they complete, and this I/O completion event may occur in the context of a separate thread previously unknown to the application. In summary, a single application event handler can easily handle error conditions arising from both exception events and failed asynchronous operations.

## Event Attributes

Symbolic Name	Description
<code>VI_ATTR_EVENT_TYPE</code>	Unique logical identifier of the event. This attribute always has the value of <code>VI_EVENT_EXCEPTION</code> for this event type.
<code>VI_ATTR_STATUS</code>	Contains the status code returned by the operation generating the error.
<code>VI_ATTR_OPER_NAME</code>	Contains the name of the operation generating the event.

---

## Related Topics

[VI\\_ATTR\\_EVENT\\_TYPE](#)

[VI\\_ATTR\\_OPER\\_NAME](#)

VI\_ATTR\_STATUS

viEnableEvent

VI\_EVENT\_GPIB\_CIC

Resource Classes

GPIB INTFC

Description

Notification that the GPIB controller has gained or lost CIC (controller in charge) status.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.
VI_ATTR_GPIB_RECV_CIC_STATE	Specifies whether the CIC status was gained or lost.

---

Related Topics

INTFC Resource

VI\_ATTR\_EVENT\_TYPE

VI\_ATTR\_GPIB\_RECV\_CIC\_STATE

VI\_EVENT\_GPIB\_LISTEN

## Resource Classes

### GPIO INTFC

## Description

Notification that the GPIO controller has been addressed to listen.

## Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.

---

## Related Topics

[INTFC Resource](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

## VI\_EVENT\_GPIO\_TALK

## Resource Classes

### GPIO INTFC

## Description

Notification that the GPIO controller has been addressed to talk.

## Event Attribute

Symbolic Name	Description
---------------	-------------

VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.
--------------------	---

Related Topics

[INTFC Resource](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

VI\_EVENT\_IO\_COMPLETION

Resources Classes

GPIB INSTR, GPIB INTFC, PXI INSTR, PXI MEMACC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI MEMACC, VXI SERVANT

Description

This event notifies the application that an asynchronous operation has completed.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event. This attribute always has the value of VI_EVENT_IO_COMPLETION for this event type.
VI_ATTR_STATUS	Contains the return code of the asynchronous I/O operation that has completed.
VI_ATTR_JOB_ID	Contains the job ID of the asynchronous operation that has

	completed.
<code>VI_ATTR_BUFFER</code>	Contains the address of the buffer that was used in the asynchronous operation.
<code>VI_ATTR_RET_COUNT/ VI_ATTR_RET_COUNT_32/ VI_ATTR_RET_COUNT_64</code>	Contains the actual number of elements that were asynchronously transferred.
<code>VI_ATTR_OPER_NAME</code>	Contains the name of the operation generating the event.

---

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[MEMACC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[VI\\_ATTR\\_BUFFER](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

[VI\\_ATTR\\_JOB\\_ID](#)

[VI\\_ATTR\\_OPER\\_NAME](#)

[VI\\_ATTR\\_RET\\_COUNT/VI\\_ATTR\\_RET\\_COUNT\\_32/VI\\_ATTR\\_RET\\_COUNT\\_64](#)

VI\_ATTR\_STATUS

VI\_EVENT\_PXI\_INTR

Resource Classes

PXI INSTR

Description

This event notifies that a PXI interrupt has occurred.

Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.
VI_ATTR_PXI_RECV_INTR_SEQ	The index of the interrupt sequence that detected the interrupt condition.
VI_ATTR_PXI_RECV_INTR_DATA	The first PXI/PCI register that was read in the successful interrupt detection sequence.

---

Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

[VI\\_ATTR\\_PXI\\_RECV\\_INTR\\_DATA](#)

[VI\\_ATTR\\_PXI\\_RECV\\_INTR\\_SEQ](#)

VI\_EVENT\_SERVICE\_REQ

## Resource Classes

GPIB INSTR, GPIB INTFC, TCPIP INSTR, USB INSTR, VXI INSTR

## Description

This event notifies the application that a service request was received from the device or interface associated with the given session.



**Note** When you receive a `VI_EVENT_SERVICE_REQ` on an instrument session, you must call `viReadSTB()` to guarantee delivery of future service request events on the given session.

## Event Attributes

Symbolic Name	Description
<code>VI_ATTR_EVENT_TYPE</code>	Unique logical identifier of the event. This attribute always has the value of <code>VI_EVENT_SERVICE_REQ</code> for this event type.

---

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

[viReadSTB](#)

## VI\_EVENT\_TRIG

# Resource Classes

GPIB INTFC, VXI INSTR, VXI BACKPLANE, VXI SERVANT

## Description

This event notifies the application that a trigger interrupt was received from the device. This may be either a hardware or software trigger, depending on the interface and the current session settings.

## Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event. This attribute always has the value of VI_EVENT_TRIG for this event type.
VI_ATTR_RECV_TRIG_ID	The identifier of the triggering mechanism on which the specified trigger event was received.

---

## Related Topics

[BACKPLANE Resource](#)

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

[VI\\_ATTR\\_RECV\\_TRIG\\_ID](#)

[VI\\_ATTR\\_TRIG\\_ID](#)



# VI\_EVENT\_USB\_INTR

## Resource Classes

USB INSTR, USB RAW

## Description

This event notifies that a USB interrupt has occurred.

## Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.
VI_ATTR_STATUS	Contains the status code returned by this event.
VI_ATTR_USB_RECV_INTR_SIZE	The number of bytes of USB interrupt data that is stored.
VI_ATTR_USB_RECV_INTR_DATA	The actual received data from the USB Interrupt.

---

## Related Topics

[INSTR Resource](#)

[RAW Resource](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

[VI\\_ATTR\\_STATUS](#)

[VI\\_ATTR\\_USB\\_RECV\\_INTR\\_DATA](#)

[VI\\_ATTR\\_USB\\_RECV\\_INTR\\_SIZE](#)

# VI\_EVENT\_VXI\_SIGP

## Resource Classes

VXI INSTR

## Description

This event notifies the application that a VXIbus signal or VXIbus interrupt was received from the device associated with the given session.

## Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event. This attribute always has the value of VI_EVENT_VXI_SIGP for this event type.
VI_ATTR_SIGP_STATUS_ID	The 16-bit Status/ID value retrieved during the IACK cycle or from the Signal register.

---

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

[VI\\_ATTR\\_SIGP\\_STATUS\\_ID](#)

# VI\_EVENT\_VXI\_VME\_INTR

## Resource Classes

VXI INSTR

## Description

This event notifies the application that a VXIbus interrupt was received from the device associated with the given session.

## Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event. This attribute always has the value of VI_EVENT_VXI_VME_INTR for this event type.
VI_ATTR_INTR_STATUS_ID	The 32-bit Status/ID value retrieved during the IACK cycle.
VI_ATTR_RECV_INTR_LEVEL	The VXI interrupt level on which the interrupt was received.

---

## Related Topics

[INSTR Resource](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

[VI\\_ATTR\\_INTR\\_STATUS\\_ID](#)

[VI\\_ATTR\\_RECV\\_INTR\\_LEVEL](#)

# VI\_EVENT\_VXI\_VME\_SYSFAIL

## Resource Classes

VXI BACKPLANE

## Description

Notification that the VXI/VME SYSFAIL\* line has been asserted.

## Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event.

---

## Related Topics

[BACKPLANE Resource](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

# VI\_EVENT\_VXI\_VME\_SYSRESET

## Resource Classes

VXI BACKPLANE, VXI SERVANT

## Description

Notification that the VXI/VME SYSRESET\* line has been asserted.

# Event Attributes

Symbolic Name	Description
VI_ATTR_EVENT_TYPE	Unique logical identifier of the event

---

## Related Topics

[BACKPLANE Resource](#)

[SERVANT Resource](#)

[VI\\_ATTR\\_EVENT\\_TYPE](#)

# Operations

These topics describe the VISA operations. The operation descriptions are listed in alphabetical order for easy reference.

Each event description contains a brief **Purpose** statement below the title. You will then see the operation defined in both ANSI C and Visual Basic version 4 syntax, with the parameters set in **boldface** type. A list indicating the supported resource classes, such as GPIB, Serial, etc. is followed by a table that describes each parameter and indicates whether it is an input or output parameter (or both, in some cases). The **Return Values** section describes the completion and error codes, followed by a detailed **Description** section. The **Related Items** section directs you toward related operations, attributes, events, or resource descriptions. If you want to know specifically about attributes, events, and operations of the INSTR Resource, for example, you should navigate to the [INSTR Resource](#) topic.

## viAssertIntrSignal

### Purpose

Asserts the specified interrupt or signal.

### C Syntax

```
ViStatus viAssertIntrSignal(ViSession vi, ViInt16 mode, ViUInt32 statusID)
```

### Visual Basic Syntax

```
viAssertIntrSignal&(ByVal vi&, ByVal mode%, ByVal statusID&)
```

## Resource Classes

VXI BACKPLANE, VXI SERVANT

### Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>mode</b>	IN	This specifies how to assert the interrupt. Refer to the Description section for actual values.
<b>statusID</b>	IN	This is the status value to be presented during an interrupt acknowledge cycle.

### Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the

	resource identified by <b>vi</b> has been locked for this kind of access.
<code>VI_ERROR_BERR</code>	Bus error occurred during transfer.
<code>VI_ERROR_INTR_PENDING</code>	An interrupt is still pending from a previous call.
<code>VI_ERROR_INV_MODE</code>	The value specified by the <b>mode</b> parameter is invalid.
<code>VI_ERROR_NSUP_INTR</code>	The interface cannot generate an interrupt on the requested level or with the requested <b>statusID</b> value.
<code>VI_ERROR_NSUP_MODE</code>	The specified <b>mode</b> is not supported by this VISA implementation.

## Description

This operation can be used to assert a device interrupt condition. In VXI, for example, this can be done with either a VXI signal or a VXI interrupt. On certain bus types, the **statusID** parameter may be ignored. The following table lists the valid values for the **mode** parameter.

Mode	Action Description
<code>VI_ASSERT_USE_ASSIGNED</code>	Use whatever notification method that has been assigned to the local device.
<code>VI_ASSERT_SIGNAL</code>	Send the notification via a VXI signal.



<p>VI_ASSERT_IRQ1 - VI_ASSERT_IRQ7</p>	<p>Send the interrupt via the specified VXI/VME IRQ line. This uses the standard VXI/VME ROAK (Release On Acknowledge) interrupt mechanism, rather than the older VME RORA (Release On Register Access) mechanism.</p>
--	--

## Related Topics

[BACKPLANE Resource](#)

[SERVANT Resource](#)

[viAssertUtilSignal](#)

## viAssertTrigger

### Purpose

Asserts software or hardware trigger.

### C Syntax

```
ViStatus viAssertTrigger(ViSession vi, ViUInt16 protocol)
```

### Visual Basic Syntax

```
viAssertTrigger&(ByVal vi&, ByVal protocol%)
```

### Resource Classes

GPIO INSTR, GPIO INTRFC, PXI INSTR, PXI BACKPLANE, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI BACKPLANE

## Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
protocol	IN	<p>Trigger protocol to use during assertion. Valid values are:</p> <p><b>Range</b></p> <p><b>GPIB, Serial, TCPIP, USB</b></p> <p>VI_TRIG_PROT_DEFAULT (0)</p> <p><b>VXI</b></p> <p>VI_TRIG_PROT_DEFAULT (0), VI_TRIG_PROT_ON (1), VI_TRIG_PROT_OFF (2), and VI_TRIG_PROT_SYNC (5)</p> <p><b>PXI</b></p> <p>VI_TRIG_PROT_RESERVE (6) VI_TRIG_PROT_UNRESERVE (7)</p>

## Return Values

Completion Codes	Description
VI_SUCCESS	The specified trigger was successfully asserted to the device.

Error Codes	Description
-------------	-------------

VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_INV_PROT	The protocol specified is invalid.
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_LINE_IN_USE	The specified trigger line is currently in use.
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 unasserted).
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_CONN_LOST	The I/O connection for the given session has been lost.

## Description

The `viAssertTrigger()` operation sources a software or hardware trigger dependent on the interface type.

### Software Triggers for 488.2 Instruments (GPIB, VXI, TCPIP, and USB)

This operation sends an IEEE-488.2 software trigger to the addressed device. For software triggers, `VI_TRIG_PROT_DEFAULT` is the only valid protocol. The bus-specific details are:

- For a GPIB device, VISA addresses the device to listen and then sends the GPIB GET command.
- For a VXI device, VISA sends the Word Serial Trigger command.
- For a USB device, VISA sends the TRIGGER message ID on the Bulk-OUT pipe.

### Software Triggers for Non-488.2 Instruments (Serial INSTR, TCPIP SOCKET, and USB RAW)

If `VI_ATTR_IO_PROT` is `VI_PROT_4882_STRS`, this operations sends `"*TRG\n"` to the device; otherwise, this operation is not valid. For software triggers, `VI_TRIG_PROT_DEFAULT` is the only valid protocol.

### Hardware Triggering for VXI

For hardware triggers to VXI instruments, `VI_ATTR_TRIG_ID` must first be set to the desired trigger line to use; this operation performs the specified trigger operation on

the previously selected trigger line. For VXI hardware triggers, `VI_TRIG_PROT_DEFAULT` is equivalent to `VI_TRIG_PROT_SYNC`.

## Trigger Reservation for PXI

For PXI instruments, this operation reserves or releases (unreserves) a trigger line for use in external triggering. For PXI triggers, `VI_TRIG_PROT_RESERVE` and `VI_TRIG_PROT_UNRESERVE` are the only valid protocols.

---

## Related Topics

[BACKPLANE Resource](#)

[INSTR Resource](#)

[INTFC Resource](#)

[SOCKET Resource](#)

[VI\\_ATTR\\_TRIG\\_ID](#)

## viAssertUtilSignal

### Purpose

Asserts or deasserts the specified utility bus signal.

### C Syntax

```
viStatus viAssertUtilSignal(ViSession vi, ViUInt16 line)
```

### Visual Basic Syntax

```
viAssertUtilSignal& (ByVal vi&, ByVal line%)
```

## Resource Classes

VXI BACKPLANE, VXI SERVANT

### Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>line</b>	IN	Specifies the utility bus signal to assert. This can be the value VI_UTIL_ASSERT_SYSRESET, VI_UTIL_ASSERT_SYSFAIL, or VI_UTIL_DEASSERT_SYSFAIL.

### Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.

VI_ERROR_TMO	Timeout expired before operation completedS.
VI_ERROR_INV_LINE	The value specified by the <b>line</b> parameter is invalid.

## Description

This operation can be used to assert either the SYSFAIL or SYSRESET utility bus interrupts on the VXIbus backplane. This operation is valid only on BACKPLANE (mainframe) and VXI SERVANT (servant) sessions.

Asserting SYSRESET (also known as HARD RESET in the VXI specification) should be used only when it is necessary to promptly terminate operation of all devices in a VXIbus system. This is a serious action that always affects the entire VXIbus system.

---

## Related Topics

[BACKPLANE Resource](#)

[SERVANT Resource](#)

[viAssertIntrSignal](#)

## viBufRead

### Purpose

Reads data from device or interface through the use of a formatted I/O read buffer.

### C Syntax

```
ViStatus viBufRead(ViSession vi, ViPBuf buf, ViUInt32 count, ViPUIInt32 retCount)
```

## Visual Basic Syntax

```
viBufRead&(ByVal vi&, ByVal buf$, ByVal count&, retCount&)
```

## Resource Classes

GPIB INSTR, GPIB INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>buf</b>	OUT	Location of a buffer to receive data from device.
<b>count</b>	IN	Number of bytes to be read.
<b>retCount</b>	OUT	Number of bytes actually transferred.

## Return Values

Completion Codes	Description
VI_SUCCESS	The operation completed successfully and the END indicator was received (for interfaces that have END indicators). This completion code is returned regardless of whether the termination character is received or the number of bytes read is equal to <b>count</b> .
VI_SUCCESS_TERM_CHAR	The specified termination character was read but no END



	indicator was received. This completion code is returned regardless of whether the number of bytes read is equal to <b>count</b> .
<code>VI_SUCCESS_MAX_CNT</code>	The number of bytes read is equal to <b>count</b> . No END indicator was received and no termination character was read.

Error Codes	Description
<code>VI_ERROR_INV_OBJECT</code>	The given session reference is invalid.
<code>VI_ERROR_NSUP_OPER</code>	The given <b>vi</b> does not support this operation.
<code>VI_ERROR_RSRC_LOCKED</code>	Specified operation could not be performed because the resource identified by <b>vi</b> has been locked for this kind of access.
<code>VI_ERROR_TMO</code>	Timeout expired before operation completed.
<code>VI_ERROR_IO</code>	An unknown I/O error occurred during transfer.

## Description

The `viBufRead()` operation is similar to `viRead()` and does not perform any kind of data formatting. It differs from `viRead()` in that the data is read from the formatted I/O read buffer—the same buffer used by `viScanf()` and related operations—rather than directly from the device. You can intermix this operation with `viScanf()`, but you should not mix it with `viRead()`.

`VI_NULL` is a special value for the **retCount** parameter. If you pass `VI_NULL` for **retCount**, the number of bytes transferred is not returned. You may find this useful if you need to know only whether the operation succeeded or failed.



**Note** The **retCount** and **buf** parameters always are valid on both success and error.

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[viBufWrite](#)

[viRead](#)

## viBufWrite

### Purpose

Writes data to a formatted I/O write buffer synchronously.

### C Syntax

```
ViStatus viBufWrite(ViSession vi, ViBuf buf, ViUInt32 count, ViUInt32 retCount)
```

### Visual Basic Syntax

```
viBufWrite&(ByVal vi&, ByVal buf$, ByVal count&, retCount&)
```

## Resource Classes

GPIO INSTR, GPIO INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>buf</b>	IN	Location of a block of data.
<b>count</b>	IN	Number of bytes to be written.
<b>retCount</b>	OUT	Number of bytes actually transferred.

## Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given vi does not support this operation.


<code>VI_ERROR_RSRC_LOCKED</code>	Specified operation could not be performed because the resource identified by <code>vi</code> has been locked for this kind of access.
<code>VI_ERROR_TMO</code>	Timeout expired before operation completed.
<code>VI_ERROR_INV_SETUP</code>	Unable to start write operation because setup is invalid (due to attributes being set to an inconsistent state).
<code>VI_ERROR_IO</code>	An unknown I/O error occurred during transfer.

## Description

The `viBufWrite()` operation is similar to `viWrite()` and does not perform any kind of data formatting. It differs from `viWrite()` in that the data is written to the formatted I/O write buffer—the same buffer used by `viPrintf()` and related operations—rather than directly to the device. You can intermix this operation with `viPrintf()`, but you should not mix it with `viWrite()`.

If this operation returns `VI_ERROR_TMO`, the write buffer for the specified session is cleared.

`VI_NULL` is a special value for the **retCount** parameter. If you pass `VI_NULL` for **retCount**, the number of bytes transferred is not returned. You may find this useful if you need to know only whether the operation succeeded or failed.

 **Note** The **retCount** parameter always is valid on both success and error.

---

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[viBufRead](#)

[viWrite](#)

# viClear

## Purpose

Clears a device.

## C Syntax

```
ViStatus viClear(ViSession vi)
```

## Visual Basic Syntax

```
viClear&(ByVal vi&)
```

## Resource Classes

GPIB INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR

## Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.

## Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
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_BERR	Bus error occurred during transfer.
VI_ERROR_NCIC	The interface associated with the given vi is not currently the controller in charge.

<code>VI_ERROR_NLISTENERS</code>	No-listeners condition is detected (both <code>NRFD</code> and <code>NDAC</code> are unasserted).
<code>VI_ERROR_INV_SETUP</code>	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).
<code>VI_ERROR_CONN_LOST</code>	The I/O connection for the given session has been lost.

## Description

The `viClear()` operation clears the device input and output buffers. The bus-specific details are:

### Clear for 488.2 Instruments (GPIB, VXI, TCPIP, and USB)

- For a GPIB device, VISA sends the Selected Device Clear command.
- For a VXI device, VISA sends the Word Serial Clear command.
- For a USB device, VISA sends the `INITIATE_CLEAR` and `CHECK_CLEAR_STATUS` commands on the control pipe.

### Clear for Non-488.2 Instruments (Serial INSTR, TCPIP SOCKET, and USB RAW)

- For Serial INSTR sessions, VISA flushes (discards) the I/O output buffer, sends a break, and then flushes (discards) the I/O input buffer.
- For TCPIP SOCKET sessions, VISA flushes (discards) the I/O buffers.
- For USB RAW sessions, VISA resets the endpoints referred to by the attributes `VI_ATTR_USB_BULK_IN_PIPE` and `VI_ATTR_USB_BULK_OUT_PIPE`.

Invoking `viClear()` also discards the read and write buffers used by the formatted I/O services for that session.

---

## Related Topics

[INSTR Resource](#)

[SOCKET Resource](#)

# viClose

## Purpose

Closes the specified session, event, or find list.

## C Syntax

```
ViStatus viClose(ViObject vi)
```

## Visual Basic Syntax

```
viClose& (ByVal vi&)
```

## Resource Classes

All I/O session types, all event object types, VISA Resource Manager

## Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session, event, or find list.

## Return Values

Completion Codes	Description
VI_SUCCESS	Session closed successfully.



<code>VI_WARN_NULL_OBJECT</code>	The specified object reference is uninitialized.
----------------------------------	--

Error Codes	Description
<code>VI_ERROR_INV_OBJECT</code>	The given object reference is invalid.
<code>VI_ERROR_CLOSING_FAILED</code>	Unable to deallocate the previously allocated data structures corresponding to this session or object reference.

## Description

The `viClose()` operation closes a session, event, or a find list. In this process all the data structures that had been allocated for the specified `vi` are freed. Calling `viClose()` on a VISA Resource Manager session will also close all I/O sessions associated with that resource manager session.

---

## Related Topics

[viFindRsrc](#)

[viOpen](#)

[viOpenDefaultRM](#)

[VISA Resource Template](#)

[viWaitOnEvent](#)

[viDisableEvent](#)

# Purpose

Disables notification of the specified event type(s) via the specified mechanism(s).

# C Syntax

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

# Visual Basic Syntax

```
viDisableEvent&(ByVal vi&, ByVal eventType&, ByVal mechanism%)
```

# Resource Classes

All I/O session types

# Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>eventType</b>	IN	Logical event identifier.
<b>mechanism</b>	IN	Specifies event handling mechanisms to be disabled. The queuing mechanism is disabled by specifying <code>VI_QUEUE (1)</code> , and the callback mechanism is disabled by specifying <code>VI_HNDLR (2)</code> or <code>VI_SUSPEND_HNDLR (4)</code> . It is possible to disable both mechanisms simultaneously by specifying <code>VI_ALL_MECH (FFFFh)</code> .

## Return Values

Completion Codes	Description
<code>VI_SUCCESS</code>	Event disabled successfully.
<code>VI_SUCCESS_EVENT_DIS</code>	Specified event is already disabled for at least one of the specified mechanisms.

Error Codes	Description
<code>VI_ERROR_INV_OBJECT</code>	The given session reference is invalid.
<code>VI_ERROR_INV_EVENT</code>	Specified <b>eventType</b> is not supported by the resource.
<code>VI_ERROR_INV_MECH</code>	Invalid <b>mechanism</b> specified.

## Description

The `viDisableEvent()` operation disables servicing of an event identified by the **eventType** parameter for the mechanisms specified in the **mechanism** parameter. This operation prevents **new** event occurrences from being added to the queue(s). However, event occurrences already existing in the queue(s) are not flushed. Use `viDiscardEvents()` if you want to discard events remaining in the queue(s).

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.



**Note** Calling `viDisableEvent()` prevents future events from being raised on the given session. When the method returns to the application, it is possible that a callback may still be active, such as on another thread. It is valid for a user to call `viDisableEvent()` from within a callback, but this is not recommended.

## Related Topics

[viEnableEvent](#)

[VISA Resource Template](#)

[viUninstallHandler](#)

## viDiscardEvents

### Purpose

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

### C Syntax

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

### Visual Basic Syntax

```
viDiscardEvents&(ByVal vi&, ByVal eventType&, ByVal mechanis  
m%)
```

### Resource Classes

All I/O session types

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>eventType</b>	IN	Logical event identifier.
<b>mechanism</b>	IN	Specifies the mechanisms for which the events are to be discarded. The <code>VI_QUEUE</code> (1) value is specified for the queuing mechanism and the <code>VI_SUSPEND_HNDLR</code> (4) value is specified for the pending events in the callback mechanism. It is possible to specify both mechanisms simultaneously by specifying <code>VI_ALL_MECH</code> (FFFFh).

## Return Values

Completion Codes	Description
<code>VI_SUCCESS</code>	Event queue flushed successfully.
<code>VI_SUCCESS_QUEUE_EMPTY</code>	Operation completed successfully, but queue was already empty.

Error Codes	Description
<code>VI_ERROR_INV_OBJECT</code>	The given session reference is invalid.
<code>VI_ERROR_INV_EVENT</code>	Specified <b>eventType</b> is not supported by the resource.

VI_ERROR_INV_MECH	Invalid <b>mechanism</b> specified.
-------------------	-------------------------------------

## Description

The `viDiscardEvents()` operation discards all pending occurrences of the specified event types and mechanisms from the specified session. Specifying `VI_ALL_ENABLED_EVENTS` for the **eventType** parameter discards events of every type that is enabled for the given session. The information about all the event occurrences which have not yet been handled is discarded. This operation is useful to remove event occurrences that an application no longer needs. The discarded event occurrences are not available to a session at a later time. This operation does not apply to event contexts that have already been delivered to the application.

---

## Related Topics

[viDisableEvent](#)

[viEnableEvent](#)

[VISA Resource Template](#)

[viWaitOnEvent](#)

## viEnableEvent

### Purpose

Enables notification of a specified event.

### C Syntax

```
ViStatus viEnableEvent(ViSession vi, ViEventType eventType, Vi
UInt16 mechanism, ViEventFilter context)
```

## Visual Basic Syntax

```
viEnableEvent&(ByVal vi&, ByVal eventType&, ByVal mechanism%,  
ByVal context&)
```

## Resource Classes

All I/O session types

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>eventType</b>	IN	Logical event identifier.
<b>mechanism</b>	IN	Specifies event handling mechanisms to be enabled. The queuing mechanism is enabled by specifying <code>VI_QUEUE (1)</code> , and the callback mechanism is enabled by specifying <code>VI_HNDLR (2)</code> or <code>VI_SUSPEND_HNDLR (4)</code> .
<b>context</b>	IN	<code>VI_NULL (0)</code> .

## Return Values

Completion Codes	Description
<code>VI_SUCCESS</code>	Event enabled successfully.
<code>VI_SUCCESS_EVENT_EN</code>	Specified event is already enabled for at least one of the specified mechanisms.

Error Codes	Description
<code>VI_ERROR_INV_OBJECT</code>	The given session reference is invalid.
<code>VI_ERROR_INV_EVENT</code>	Specified <b>eventType</b> is not supported by the resource.
<code>VI_ERROR_INV_MECH</code>	Invalid <b>mechanism</b> specified for the event.
<code>VI_ERROR_INV_CONTEXT</code>	Specified event context is invalid.
<code>VI_ERROR_INV_SETUP</code>	Unable to start write operation because setup is invalid (due to attributes being set to an inconsistent state).
<code>VI_ERROR_HNDLR_NINSTALLED</code>	A handler is not currently installed for the specified event. The session cannot be enabled for the <code>VI_HNDLR</code> mode of the callback mechanism.
<code>VI_ERROR_NSUP_MECH</code>	The specified <b>mechanism</b> is not supported for the given <b>eventType</b> .

## Description

The `viEnableEvent()` 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 which have previously been enabled on this session, making it easier to switch between the two callback mechanisms for multiple events.

NI-VISA does not support enabling both the queue and the handler for the same event type on the same session. If you need to use both mechanisms for the same event type, you should open multiple sessions to the resource.

## Related Topics

[Events](#)

[viDisableEvent](#)

[viEventHandler](#)

[viInstallHandler](#)

[VISA Resource Template](#)

[viUninstallHandler](#)

[viWaitOnEvent](#)

## viEventHandler

### Purpose

Event service handler procedure prototype.

### C Syntax

```
ViStatus _VI_FUNCH viEventHandler(ViSession vi, ViEventType
```

**eventType**, ViEvent **context**, ViAddr **userHandle**)

Visual Basic Syntax

N/A

Resource Classes

All I/O session types

Parameters

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

Return Values

Completion Codes	Description
VI_SUCCESS	Event handled successfully.
VI_SUCCESS_NCHAIN	Event handled successfully. Do not invoke any other handlers on this session for this event.

## Description

`viEventHandler()` is not an actual VISA operation. Rather, it is the prototype for a user event handler that is installed with the `viInstallHandler()` operation. The user handler is called whenever a session receives an event and is enabled for handling events in the `VI_HNDLR` mode. The handler services the event and returns `VI_SUCCESS` on completion. The VISA system automatically invokes the `viClose()` operation on the event context when a user handler returns.

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.



**Note** For advanced users—If the user handler will not return to VISA, the application should call `viClose()` on the event context to manually delete the event object. This situation may occur when a handler throws a C++ exception in response to a VISA exception event.

Normally, an application should always return `VI_SUCCESS` from all callback handlers. If a specific handler does not want other handlers to be invoked for the given event for the given session, it should return `VI_SUCCESS_NCHAIN`. No return value from a handler on one session will affect callbacks on other sessions. Future versions of VISA (or specific implementations of VISA) may take actions based on other return values, so a user should return `VI_SUCCESS` from handlers unless there is a specific reason to do otherwise.

---

## Related Topics

[`viInstallHandler`](#)

[VISA Resource Template](#)

[`viUninstallHandler`](#)

## `viFindNext`

## Purpose

Returns the next resource from the list of resources found during a previous call to `viFindRsrc()`.

## C Syntax

```
ViStatus viFindNext(ViFindList findList, ViChar instrDesc[])
```

## Visual Basic Syntax

```
viFindNext&(ByVal findList&, ByVal instrDesc$)
```

## Resource Classes

VISA Resource Manager

## Parameters

Name	Direction	Description
findList	IN	Describes a find list. This parameter must be created by <code>viFindRsrc()</code> .
instrDesc	OUT	Returns a string identifying the location of a device. Strings can then be passed to <code>viOpen()</code> to establish a session to the given device.

## Return Values

Completion Codes	Description
VI_SUCCESS	Resource(s) found.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given object reference is invalid.
VI_ERROR_NSUP_OPER	The given <b>findList</b> does not support this operation.
VI_ERROR_RSRC_NFOUND	There are no more matches.

## Description

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



**Note** The size of the `instrDesc` parameter should be ***at least*** 256 bytes.

## Related Topics

[viFindRsrc](#)

[VISA Resource Template](#)

## viFindRsrc

### Purpose

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

## C Syntax

```
ViStatus viFindRsrc(ViSession sesn, ViString expr, ViPFindList findList, ViPUInt32 retcnt, ViChar instrDesc[])
```

## Visual Basic Syntax

```
viFindRsrc&(ByVal sesn&, ByVal expr$, findList&, retcnt&, ByVal instrDesc$)
```

## Resource Classes

### VISA Resource Manager

## Parameters

Name	Direction	Description
<b>sesn</b>	IN	Resource Manager session (should always be the session returned from <code>viOpenDefaultRM()</code> ).
<b>expr</b>	IN	This is a regular expression followed by an optional logical expression. Refer to the discussion of the Description String in the <b>Description</b> section of this operation.
<b>findList</b>	OUT	Returns a handle identifying this search session. This handle will be used as an input in <code>viFindNext()</code> .
<b>retcnt</b>	OUT	Number of matches.
<b>instrDesc</b>	OUT	Returns a string identifying the location of a device. Strings can then be passed to <code>viOpen()</code> to establish a session to the given device.

## Return Values

Completion Codes	Description
VI_SUCCESS	Resource(s) found.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given <b>sesn</b> does not support this operation. This operation is supported only by a Resource Manager session.
VI_ERROR_INV_EXPR	Invalid expression specified for search.
VI_ERROR_RSRC_NFOUND	Specified expression does not match any devices.

## Description

The `viFindRsrc()` operation matches the value specified in the **expr** parameter with the resources available for a particular interface. A regular expression is a string consisting of ordinary characters as well as special characters. You use a regular expression to specify patterns to match in a given string; in other words, it is a search criterion. The `viFindRsrc()` operation uses a case-insensitive compare feature when matching resource names against the regular expression specified in **expr**. For example, calling `viFindRsrc()` with `"VXI?*INSTR"` would return the same resources as invoking it with `"vxi?*instr"`.

On successful completion, this function returns the first resource found (**instrDesc**) and returns a count (**retcnt**) to indicate if there were more resources found for the

designated interface. This function also returns, in the **findList** parameter, 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()`. Notice that **retcnt** and **findList** are optional parameters. This is useful if only the first match is important, and the number of matches is not needed. If you specify `VI_NULL` in the **findList** parameter and the operation completes successfully, VISA automatically invokes `viClose()` on the find list handle rather than returning it to the application.



**Note** The size of the `instrDesc` parameter should be **at least** 256 bytes.



**Note** All resource strings returned by `viFindRsrc()` will always be recognized by `viOpen()`. However, `viFindRsrc()` will not necessarily return all strings that you can pass to `viParseRsrc()` or `viOpen()`. This is especially true for network and TCPIP resources. If a resource does not appear in the list, you can explicitly add it in the NI-VISA configuration utility (MAX on Windows, `visaconf` on UNIX), and then `viFindRsrc()` will return it. The configuration utility also has other options that expand or limit the set of resources that `viFindRsrc()` returns.

The search criteria specified in the **expr** parameter has two parts: a regular expression over a resource string, 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.

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 ( <code>\?</code> ),



	it matches the ? character instead of any one character.
[list]	Matches any one character from the enclosed list. You can use a hyphen to match a range of characters.
[^list]	Matches any character not in the enclosed list. You can use a hyphen 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 VX(I G)PIB.
(exp)	Grouping characters or expressions.

Regular Expression	Sample Matches
GPIB?*INSTR	Matches GPIB0::2::INSTR, and GPIB1::1::1::INSTR.
GPIB[0-9]*::?*INSTR	Matches GPIB0::2::INSTR and GPIB1::1::1::INSTR.
GPIB[^0]::?*INSTR	Matches GPIB1::1::1::INSTR but not GPIB0::2::INST

	R or GPIB12::8::INSTR.
VXI?*INSTR	Matches VXI0::1::INSTR.
?*VXI[0-9]*::?*INSTR	Matches VXI0::1::INSTR.
ASRL[0-9]*::?*INSTR	Matches ASRL1::INSTR but not VXI0::5::INSTR.
ASRL1+::INSTR	Matches ASRL1::INSTR and ASRL11::INSTR but not ASRL2::INSTR.
(GPIB VXI)?*INSTR	Matches GPIB1::5::INSTR and VXI0::3::INSTR but not ASRL2::INSTR.
(GPIB0 VXI0)::1::INSTR	Matches GPIB0::1::INSTR and VXI0::1::INSTR.
?*INSTR	Matches all INSTR (device) resources.
?*VXI[0-9]*::?*MEMACC	Matches VXI0::MEMACC.
VXI0::?*	Matches VXI0::1::INSTR, VXI0::2::INSTR, and VXI0::MEMACC.
?*	Matches all resources.
visa://hostname/?*	Matches all resources on the specified remote system. The

	hostname can be represented as either an IP address (dot-notation) or network machine name. This remote system need not be a configured remote system.
<code>/?*</code>	Matches all resources on the local machine. Configured remote systems are not queried.
<code>visa:/ASRL?*INSTR</code>	Matches all ASRL resources on the local machine and returns them in URL format (for example, <code>visa:/ASRL1::INSTR</code> ).

You can use the NI-VISA configuration utility (MAX on Windows, `visaconf` on UNIX) to access certain NI-VISA servers by default. All expressions without the preceding `"/` will be matched with resources on the configured remote systems.

By using the optional attribute expression, you can construct flexible and powerful expressions with the use of logical ANDs (`&&`), ORs (`||`), and NOTs (`!`). You can use equal (`==`) and unequal (`!=`) comparators to compare attributes of any type, and other inequality comparators (`>`, `<`, `>=`, `<=`) to compare attributes of numeric type. Use only global attributes in the attribute expression. Local attributes are not allowed in the logical expression part of the **expr** parameter.

Expr Parameter	Meaning
<code>GPIB[0-9]*::?*::?*::INSTR {VI_ATTR_GPIB_SECONDARY_ADDR &gt; 0 &amp;&amp; VI_ATTR_GPIB_SECONDARY_ADDR &lt; 10}</code>	Find all GPIB devices that have secondary addresses from 1 to 9.
<code>ASRL?*INSTR{VI_ATTR_ASRL_BAUD == 9600}</code>	Find all serial ports configured at 9600 baud.
<code>?*VXI?INSTR{VI_ATTR_MANF_ID == 0xFF6 &amp;&amp; !(VI_ATTR_VXI_LA == 0    VI_ATT</code>	Find all VXI instrument resources having manufacturer ID FF6 and which are not

```
R_SLOT <= 0) }
```

logical address 0, slot 0, or external controllers.

## Related Topics

[viClose](#)

[viFindNext](#)

[VISA Resource Template](#)

## viFlush

### Purpose

Manually flushes the specified buffers associated with formatted I/O operations and/or serial communication.

### C Syntax

```
ViStatus viFlush(ViSession vi, ViUInt16 mask)
```

### Visual Basic Syntax

```
viFlush&(ByVal vi&, ByVal mask%)
```

### Resource Classes

GPIO INSTR, GPIO INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>mask</b>	IN	Specifies the action to be taken with flushing the buffer. Refer to the <b>Description</b> section for more information.

## Return Values

Completion Codes	Description
VI_SUCCESS	Buffers flushed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by <b>vi</b> has been locked for this kind of access.
VI_ERROR_IO	Could not perform Read/Write operation because of I/O error.
VI_ERROR_TMO	The Read/Write operation was aborted because timeout expired while operation was in progress.

VI_ERROR_INV_MASK	The specified <b>mask</b> does not specify a valid flush operation on Read/Write resource.
-------------------	--

## Description

The value of **mask** can be one of the following flags.

Flag	Interpretation
VI_READ_BUF (1)	Discard the read buffer contents. If data was present in the read buffer and no END-indicator was present, read from the device until encountering an END indicator (which causes the loss of data). This action resynchronizes the next <code>viScanf()</code> call to read a <TERMINATED RESPONSE MESSAGE>. (Refer to the IEEE 488.2 standard.)
VI_READ_BUF_DISCARD (4)	Discard the read buffer contents (does not perform any I/O to the device).
VI_WRITE_BUF (2)	Flush the write buffer by writing all buffered data to the device.
VI_WRITE_BUF_DISCARD (8)	Discard the write buffer contents (does not perform any I/O to the device).
VI_IO_IN_BUF (16)	Discard the low-level I/O receive buffer contents (same as <code>VI_IO_IN_BUF_DISCARD</code> ).
VI_IO_IN_BUF_DISCARD (64)	Discard the low-level I/O receive buffer contents (does not perform any I/O to the device).

<code>VI_IO_OUT_BUF (32)</code>	Flush the low-level I/O transmit buffer by writing all buffered data to the device.
<code>VI_IO_OUT_BUF_DISCARD (128)</code>	Discard the low-level I/O transmit buffer contents (does not perform any I/O to the device).

It is possible to combine any of these read flags and write flags for different buffers by ORing the flags. However, combining two flags for the same buffer in the same call to `viFlush()` is illegal.

Notice that when using formatted I/O operations with a session to a Serial device or Ethernet socket, a flush of the formatted I/O buffers also causes the corresponding I/O communication buffers to be flushed. For example, calling `viFlush()` with `VI_WRITE_BUF` also flushes the `VI_IO_OUT_BUF`.

In previous versions of VISA, `VI_IO_IN_BUF` was known as `VI_ASRL_IN_BUF` and `VI_IO_OUT_BUF` was known as `VI_ASRL_OUT_BUF`.

### Implicit versus Explicit Flushing

Although you can explicitly flush the buffers by making a call to `viFlush()`, the buffers are flushed implicitly under some conditions. These conditions vary for the `viPrintf()` and `viScanf()` operations.

Flushing a write buffer immediately sends any queued data to the device. The write buffer is maintained by the `viPrintf()` operation. To explicitly flush the write buffer, you can make a call to the `viFlush()` operation with a write flag set. In addition, the write buffer is flushed automatically under the following conditions:

1. When an END-indicator character is sent (that is, the `\n` character is specified in the formatting string).
2. When the buffer is full.
3. In response to a call to `viSetBuf()` with the `VI_WRITE_BUF` flag set.

Flushing a read buffer discards the data in the read buffer. This guarantees that the

next call to a `viScanf()` (or related) operation reads data directly from the device rather than from queued data residing in the read buffer. The read buffer is maintained by the `viScanf()` operation. To explicitly flush the read buffer, you can make a call to the `viFlush()` operation with a read flag set.

Also, the formatted I/O buffers of a session to a given device are reset whenever that device is cleared. Invoking the `viClear()` operation will flush the read buffer and discard the contents of the write buffers.

---

## Related Topics

[Automatically Flushing the Formatted I/O Buffers](#)

[Controlling the Serial I/O Buffers](#)

[Formatted I/O Read and Low-Level I/O Receive Buffers](#)

[Formatted I/O Write and Low-Level I/O Transmit Buffers](#)

[INSTR Resource](#)

[INTFC Resource](#)

[Manually Flushing the Formatted I/O Buffers](#)

[Recommendations for Using the VISA Buffers](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[viSetBuf](#)

[viGetAttribute](#)



# Purpose

Retrieves the state of an attribute.

# C Syntax

```
ViStatus viGetAttribute(ViObject vi, ViAttr attribute, void * attrState)
```

# Visual Basic Syntax

```
viGetAttribute&(ByVal vi&, ByVal attribute&, attrState as Any)
```

# Resource Classes

All I/O session types, all event object types, VISA Resource Manager

# Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session, event, or find list.
<b>attribute</b>	IN	Resource attribute for which the state query is made.
<b>attrState</b>	OUT	The state of the queried attribute for a specified resource. The interpretation of the returned value is defined by the individual object.

# Return Values

Completion Codes	Description
------------------	-------------

VI_SUCCESS	Attribute retrieved successfully.
------------	-----------------------------------

Error Codes	Description
VI_ERROR_INV_OBJECT	The given object reference is invalid.
VI_ERROR_NSUP_ATTR	The specified attribute is not defined by the referenced object.

## Description

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

The output parameter **attrState** is of the type of the attribute actually being retrieved. For example, when retrieving an attribute that is defined as a `ViBoolean`, your application should pass a reference to a variable of type `ViBoolean`. Similarly, if the attribute is defined as being `ViUInt32`, your application should pass a reference to a variable of type `ViUInt32`.

---

### Related Topics

[Attributes](#)

[VISA Resource Template](#)

[viSetAttribute](#)

## viGpibCommand

# Purpose

Write GPIB command bytes on the bus.

# C Syntax

```
ViStatus viGpibCommand (ViSession vi, ViBuf buf, ViUInt32 count, ViPUInt32 retCount)
```

# Visual Basic Syntax

```
viGpibCommand&(ByVal vi&, ByVal buf$, ByVal count&, retCount&)
```

# Resource Classes

GPIB INTFC

# Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>buf</b>	IN	Buffer containing valid GPIB commands.
<b>count</b>	IN	Number of bytes to be written.
<b>retCount</b>	OUT	Number of bytes actually transferred.

# Return Values

Completion Codes	Description
------------------	-------------

VI_SUCCESS	Operation completed successfully.
------------	-----------------------------------

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_INV_SETUP	Unable to start write operation because setup is invalid (due to attributes being set to an inconsistent state).
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.

## Description

This operation attempts to write **count** number of bytes of GPIB commands to the interface bus specified by **vi**. This operation is valid only on GPIB INTFC (interface) sessions. This operation returns only when the transfer terminates.

If you pass `VI_NULL` as the **retCount** parameter to the `viGpibCommand()` operation, the number of bytes transferred will not be returned. This may be useful if it is important to know only whether the operation succeeded or failed. The command bytes contained in **buf** should be valid IEEE 488-defined Multiline Interface Messages.



**Note** The **retCount** parameter always is valid on both success and error.

---

## Related Topics

[INTFC Resource](#)

## viGpibControlATN

### Purpose

Specifies the state of the ATN line and the local active controller state.

### C Syntax

```
ViStatus viGpibControlATN(ViSession vi, ViUInt16 mode)
```

### Visual Basic Syntax

```
viGpibControlATN& (ByVal vi&, ByVal mode%)
```

## Resource Classes

### GPIB INTFC

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>mode</b>	IN	Specifies the state of the ATN line and optionally the local active controller state. See the <b>Description</b> section for actual values.

## Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given <b>vi</b> does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by <b>vi</b> has been locked for this kind of access.
VI_ERROR_NCIC	The interface associated with the given <b>vi</b> is not currently the controller in charge.

VI_ERROR_INV_MODE	The value specified by the <b>mode</b> parameter is invalid.
VI_ERROR_NSUP_MODE	The specified <b>mode</b> is not supported by this VISA implementation.

## Description

This operation asserts or deasserts the GPIB ATN interface line according to the specified mode. The mode can also specify whether the local interface should acquire or release Controller Active status. This operation is valid only on GPIB INTFC (interface) sessions. The following table lists valid values for the **mode** parameter.

Mode	Action Description
VI_GPIB_ATN_DEASSERT	Deassert ATN line. The GPIB interface corresponding to the VISA session goes to standby.
VI_GPIB_ATN_ASSERT	Assert ATN line and take control synchronously without corrupting transferred data. If a data handshake is in progress, ATN is not asserted until the handshake is complete.
VI_GPIB_ATN_DEASSERT_HANDSHAKE	Deassert ATN line, and enter shadow handshake mode. The local board participates in data handshakes as an Acceptor without actually reading the data. The GPIB interface corresponding to the VISA session goes to standby.
VI_GPIB_ATN_ASSERT_IMMEDIATE	Assert ATN line and take control asynchronously and immediately without regard for any data transfer currently in progress. Generally, this should be used only under error conditions.

It is generally not necessary to use the `viGpibControlATN()` operation in most applications. Other operations such as `viGpibCommand()` and `viGpibPassControl()` modify the ATN and/or CIC state automatically.

---

Related Topics

[INTFC Resource](#)

[viGpibControlREN](#)

# viGpibControlREN

## Purpose

Controls the state of the GPIB Remote Enable (REN) interface line, and optionally the remote/local state of the device.

## C Syntax

```
ViStatus viGpibControlREN(ViSession vi, ViUInt16 mode)
```

## Visual Basic Syntax

```
viGpibControlREN&(ByVal vi&, ByVal mode%)
```

## Resource Classes

GPIB INSTR, GPIB INTFC, USB INSTR

## Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.



<b>mode</b>	IN	Specifies the state of the REN line and optionally the device remote/local state. See the <b>Description</b> section for actual values.
-------------	----	---

## Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_NCIC	The interface associated with this session is not currently the controller in charge.
VI_ERROR_NLISTENERS	No-listeners condition is detected (both NRFD and NDAC are unasserted).
VI_ERROR_NSYS_CNTL	The interface associated with this session is not the system controller.

VI_ERROR_INV_MODE	The value specified by the <b>mode</b> parameter is invalid.
-------------------	--

## Description

The `viGpibControlREN()` operation asserts or unasserts the GPIB REN interface line according to the specified mode. The mode can also specify whether the device associated with this session should be placed in local state (before deasserting REN) or remote state (after asserting REN). This operation is valid only if the GPIB interface associated with the session specified by **vi** is currently the system controller.

The following table lists special values for the **mode** parameter.

Value	Description
VI_GPIB_REN_DEASSERT	Deassert REN line.
VI_GPIB_REN_ASSERT	Assert REN line.
VI_GPIB_REN_DEASSERT_GTL	Send the Go To Local (GTL) command and deassert REN line.
VI_GPIB_REN_ASSERT_ADDRESS	Assert REN line and address device.
VI_GPIB_REN_ASSERT_LLO	Send LLO to any devices that are addressed to listen.
VI_GPIB_REN_ASSERT_ADDRESS_LLO	Address this device and send it LLO, putting it in RWLS.
VI_GPIB_REN_ASSERT_GTL	Send the Go To Local command (GTL) to this

	device.
--	---------

Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[viGpibControlATN](#)

# viGpibPassControl

## Purpose

Tell the GPIB device at the specified address to become controller in charge (CIC).

## C Syntax

ViStatus viGpibPassControl(ViSession vi, ViUInt16 primAddr, ViUInt16 secAddr)

## Visual Basic Syntax

viGPIBPassControl& (ByVal vi&, ByVal primAddr%, ByVal sec Addr r%)

## Resource Classes

GPIB INTFC

## Parameters

Name	Direction	Description
------	-----------	-------------

<b>vi</b>	IN	Unique logical identifier to a session.
<b>primAddr</b>	IN	Primary address of the GPIB device to which you want to pass control.
<b>secAddr</b>	IN	Secondary address of the targeted GPIB device. If the targeted device does not have a secondary address, this parameter should contain the value VI_NO_SEC_ADDR.

## Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.

VI_ERROR_NCIC	The interface associated with the given <b>vi</b> is not currently the controller in charge.
VI_ERROR_NLISTENERS	No Listeners condition is detected (both <b>NRFD</b> and <b>NDAC</b> are deasserted).
VI_ERROR_IO	An unknown I/O error occurred during transfer.
VI_ERROR_INV_PARAMETER	The primary or secondary address is invalid.

## Description

This operation passes controller in charge status to the device indicated by **primAddr** and **secAddr**, and then deasserts the ATN line. This operation assumes that the targeted device has controller capability. This operation is valid only on GPIB INTFC (interface) sessions.

---

## Related Topics

[INTFC Resource](#)

# viGpibSendIFC

## Purpose

Pulse the interface clear line (IFC) for at least 100 microseconds.

## C Syntax

```
ViStatus viGpibSendIFC (ViSession vi)
```

# Visual Basic Syntax

viGpibSendIFC& (ByVal **vi**&)

## Resource Classes

### GPIB INTFC

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.

## Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given <b>vi</b> does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by <b>vi</b> has been locked for this kind of access.

VI_ERROR_NSYS_CNTL	The interface associated with this session is not the system controller.
--------------------	--

## Description

This operation asserts the IFC line and becomes controller in charge (CIC). The local board must be the system controller. This operation is valid only on GPIB INTFC (interface) sessions.

## Related Topics

[INTFC Resource](#)

## viIn8/viIn16/viIn32/viIn64, viIn8Ex/viIn16Ex/viIn32Ex/viIn64Ex

## Purpose

Reads in an 8-bit, 16-bit, 32-bit, or 64-bit value from the specified memory space and offset.

## C 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)
```

```
ViStatus viIn64(ViSession vi, ViUInt16 space, ViBusAddress offset, ViPUInt64 val64)
```

et, ViPUInt64 val64)

ViStatus viIn8Ex(ViSession vi, ViUInt16 space, ViBusAddress64 offset, ViPUInt8 val8)

ViStatus viIn16Ex(ViSession vi, ViUInt16 space, ViBusAddress64 offset, ViPUInt16 val16)

ViStatus viIn32Ex(ViSession vi, ViUInt16 space, ViBusAddress64 offset, ViPUInt32 val32)

ViStatus viIn64Ex(ViSession vi, ViUInt16 space, ViBusAddress64 offset, ViPUInt64 val64)

Visual Basic Syntax

viIn8&(ByVal vi&, ByVal space%, ByVal offset&, val8 as Byte)

viIn16&(ByVal vi&, ByVal space%, ByVal offset&, val16%)

viIn32&(ByVal vi&, ByVal space%, ByVal offset&, val32&)


Resource Classes

PXI INSTR, PXI MEMACC, VXI INSTR, VXI MEMACC

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
space	IN	Specifies the address space. Refer to the table included in the <b>Description</b> section for more information.



offset	IN	<p>Offset (in bytes) of the address or register from which to read. For <code>viInXX()</code> operations, this is a 32-bit value for 32-bit applications and a 64-bit value for 64-bit applications. For <code>viInXXEx()</code> operations, this is always a 64-bit value.</p> <div data-bbox="610 401 1468 701">  <p><b>Note</b> VISA Out and VISA In functions require the offset to begin at a value evenly divisible by the number of bytes being accessed. For example, VISA Out/In 32 requires an offset evenly divisible by 4 bytes, so valid offset values could be 0x00, 0x04, 0x08, 0x0C, 0x10, 0x14, and so on. Values other than these return an error saying the offset is not properly aligned.</p> </div>
val8, val16, val32 or val64	OUT	<p>Data read from bus (8 bits for <code>viIn8[Ex]()</code>, 16 bits for <code>viIn16[Ex]()</code>, 32 bits for <code>viIn32[Ex]()</code>, and 64 bits for <code>viIn64[Ex]()</code>).</p>

## Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given vi does not support this operation.

VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by <b>vi</b> has been locked for this kind of access.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SPACE	Invalid address <b>space</b> specified.
VI_ERROR_INV_OFFSET	Invalid <b>offset</b> specified.
VI_ERROR_NSUP_OFFSET	Specified <b>offset</b> 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 <b>offset</b> is not properly aligned for the access width of the operation.
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).

## Description

The `viInXX[Ex] ()` operations use the specified address space to read in 8, 16, 32, or 64 bits of data, respectively, from the specified **offset**. These operations do not require `viMapAddress ()` to be called prior to their invocation.

The following table lists the valid entries for specifying address **space**.

Value	Description
-------	-------------

VXI and VME	VI_A16_SPACE (1) VI_A24_SPACE (2) VI_A32_SPACE (3) VI_A64_SPACE (4)
PXI INSTR	VI_PXI_CFG_SPACE (10) VI_PXI_BAR0_SPACE (11) to VI_PXI_BAR5_SPACE (16)
PXI MEMACC	VI_PXI_ALLOC_SPACE (9)

## INSTR Specific

Notice that the **offset** parameter to these operations for an INSTR Resource is the offset address relative to the device's allocated address base for the corresponding address space that was specified. For example, if **space** specifies `VI_A16_SPACE`, then **offset** specifies the offset from the logical address base address of the specified VXI device. If **space** specifies `VI_A24_SPACE` or `VI_A32_SPACE`, then **offset** specifies the offset from the base address of the VXI device's memory space allocated by the VXI Resource Manager within VXI A24 or A32 space.

## MEMACC Specific

For a MEMACC Resource, the **offset** parameter specifies an absolute address.

---

### Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[viOut8/viOut16/viOut32/viOut64, viOut8Ex/viOut16Ex/viOut32Ex/viOut64Ex](#)

## viInstallHandler

# Purpose

Installs handlers for event callbacks.

# C Syntax

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

# Visual Basic Syntax

N/A

# Resource Classes

All I/O session types

# Parameters

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

## Return Values

Completion Codes	Description
VI_SUCCESS	Event handler installed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_INV_EVENT	Specified <b>eventType</b> is not supported by the resource.
VI_ERROR_INV_HNDLR_REF	The given <b>handler</b> reference is invalid.
VI_ERROR_HNDLR_NINSTALLED	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.

## Description

The `viInstallHandler()` operation allows applications to install handlers on sessions. The handler specified in the **handler** parameter is installed along with any 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.

VISA allows applications to install multiple handlers for an **eventType** on the same session. You can install multiple handlers through multiple invocations of the `viInstallHandler()` operation, where each invocation adds to the previous list of handlers. If more than one handler is installed for an **eventType**, each of the handlers is invoked on every occurrence of the specified event(s). VISA specifies that the

handlers are invoked in Last In First Out (LIFO) order.

---

**Related Topics**

[viEnableEvent](#)

[viEventHandler](#)

[VISA Resource Template](#)

[viUninstallHandler](#)

**viLock**

**Purpose**

Establishes an access mode to the specified resources.

**C Syntax**

```
ViStatus viLock(ViSession vi, ViAccessMode lockType, ViUInt32
timeout, ViKeyId requestedKey, ViChar accesskey[])
```

**Visual Basic Syntax**

```
viLock&(ByVal vi&, ByVal lockType&, ByVal timeout&, ByVal request
edKey$, ByVal accesskey$)
```

**Resource Classes**

All I/O session types

**Parameters**

Name	Direction	Description
------	-----------	-------------

<b>vi</b>	IN	Unique logical identifier to a session.
<b>lockType</b>	IN	Specifies the type of lock requested, either <code>VI_EXCLUSIVE_LOCK</code> (1) or <code>VI_SHARED_LOCK</code> (2).
<b>timeout</b>	IN	Absolute time period (in milliseconds) that a resource waits to get unlocked by the locking session before returning an error.
<b>requestedKey</b>	IN	This parameter is not used and should be set to <code>VI_NULL</code> when <b>lockType</b> is <code>VI_EXCLUSIVE_LOCK</code> . Refer to the <b>Description</b> section for more details about using <code>VI_SHARED_LOCK</code> .
<b>accessKey</b>	OUT	This parameter should be set to <code>VI_NULL</code> when <b>lockType</b> is <code>VI_EXCLUSIVE_LOCK</code> . When <b>lockType</b> is <code>VI_SHARED_LOCK</code> , the resource returns a unique access key for the lock if the operation succeeds. This <b>accessKey</b> can then be passed to other sessions to share the lock.

## Return Values

Completion Codes	Description
<code>VI_SUCCESS</code>	Specified access mode was acquired.
<code>VI_SUCCESS_NESTED_EXCLUSIVE</code>	Specified access mode is successfully acquired, and this session has nested exclusive locks.
<code>VI_SUCCESS_NESTED_SHARED</code>	Specified access mode is successfully acquired, and this session has nested shared locks.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified <b>lockType</b> cannot be obtained because the resource is already locked with a lock type incompatible with the lock requested.
VI_ERROR_INV_LOCK_TYPE	Specified <b>lockType</b> is not supported by this resource.
VI_ERROR_INV_ACCESS_KEY	The <b>requestedKey</b> value passed in is not a valid <b>accessKey</b> to the specified resource.
VI_ERROR_TMO	Specified <b>lockType</b> could not be obtained within the specified <b>timeout</b> period.

## Description

This operation is used to obtain a lock on the specified resource. The caller can specify the type of lock requested—exclusive or shared lock—and the length of time the operation will suspend while waiting to acquire the lock before timing out. This operation can also be used for sharing and nesting locks.

The **requestedKey** and the **accessKey** parameters apply only to shared locks. These parameters are not applicable when using the lock type `VI_EXCLUSIVE_LOCK`; in this case, **requestedKey** and **accessKey** should be set to `VI_NULL`. VISA allows user applications to specify a key to be used for lock sharing, through the use of the **requestedKey** parameter. Alternatively, a user application can pass `VI_NULL` for the **requestedKey** parameter when obtaining a shared lock, in which case VISA will generate a unique access key and return it through the **accessKey** parameter. If a user application does specify a **requestedKey** value, VISA will try to use this value for the



**accessKey**. As long as the resource is not locked, VISA will use the **requestedKey** as the access key and grant the lock. When the operation succeeds, the **requestedKey** will be copied into the user buffer referred to by the **accessKey** parameter.



**Note** If requesting a `VI_SHARED_LOCK`, the size of the **accessKey** parameter should be **at least** 256 bytes.

The session that gained a shared lock can pass the **accessKey** to other sessions for the purpose of sharing the lock. The session wanting to join the group of sessions sharing the lock can use the key as an input value to the **requestedKey** parameter. VISA will add the session to the list of sessions sharing the lock, as long as the **requestedKey** value matches the **accessKey** value for the particular resource. The session obtaining a shared lock in this manner will then have the same access privileges as the original session that obtained the lock.

It is also possible to obtain nested locks through this operation. To acquire nested locks, invoke the `viLock()` operation with the same lock type as the previous invocation of this operation. For each session, `viLock()` and `viUnlock()` share a lock count, which is initialized to 0. Each invocation of `viLock()` for the same session (and for the same **lockType**) increases the lock count. In the case of a shared lock, it returns with the same **accessKey** every time. When a session locks the resource a multiple number of times, it is necessary to invoke the `viUnlock()` operation an equal number of times in order to unlock the resource. That is, the lock count increments for each invocation of `viLock()`, and decrements for each invocation of `viUnlock()`. A resource is actually unlocked only when the lock count is 0.

The VISA locking mechanism enforces arbitration of accesses to resources on an individual basis. If a session locks a resource, operations invoked by other sessions to the same resource are serviced or returned with a locking error, depending on the operation and the type of lock used. If a session has an exclusive lock, other sessions cannot modify global attributes or invoke operations, but can still get attributes and set local attributes. If the session has a shared lock, other sessions that have shared locks can also modify global attributes and invoke operations. Regardless of which type of lock a session has, if the session is closed without first being unlocked, VISA automatically performs a `viUnlock()` on that session.

The locking mechanism works for all processes and resources existing on the same

computer. When using remote resources, however, the networking protocol may not provide the ability to pass lock requests to the remote device or resource. In this case, locks will behave as expected from multiple sessions on the same computer, but not necessarily on the remote device. For example, when using the VXI-11 protocol, exclusive lock requests can be sent to a device, but shared locks can only be handled locally.

---

## Related Topics

[VISA Resource Template](#)

[viUnlock](#)

## viMapAddress/viMapAddressEx

### Purpose

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

### C Syntax

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

```
ViStatus viMapAddressEx(ViSession vi, ViUInt16 mapSpace, ViBusAddress64 mapBase, ViBusSize mapSize, ViBoolean access, ViAddr suggested, ViPAddr address)
```

### Visual Basic Syntax

```
viMapAddress&(ByVal vi&, ByVal mapSpace%, ByVal mapBase&, ByVal mapSize&, ByVal access%, ByVal suggested&, address&)
```

## Resource Classes

PXI INSTR, PXI MEMACC, VXI INSTR, VXI MEMACC

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>mapSpace</b>	IN	Specifies the address space to map. Refer to the <b>Description</b> section for more information.
<b>mapBase</b>	IN	Offset (in bytes) of the memory to be mapped. Refer to the <b>Description</b> section for more information. For <code>viMapAddress()</code> , this is a 32-bit value for 32-bit applications and a 64-bit value for 64-bit applications. For <code>viMapAddressEx()</code> , this is always a 64-bit value.
<b>mapSize</b>	IN	Amount of memory to map (in bytes).
<b>access</b>	IN	<code>VI_FALSE</code> (0).
<b>suggested</b>	IN	If <b>suggested</b> parameter is not <code>VI_NULL</code> (0), the operating system attempts to map the memory to the address specified in <b>suggested</b> . There is no guarantee, however, that the memory will be mapped to that address. This operation may map the memory into an address region different from <b>suggested</b> .
<b>address</b>	OUT	Address in your process space where the memory was mapped.

## Return Values

Completion Codes	Description
VI_SUCCESS	Mapping successful.


Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_INV_SPACE	Invalid address space specified.
VI_ERROR_INV_OFFSET	Invalid offset specified.
VI_ERROR_NSUP_OFFSET	Specified region is not accessible from this hardware.
VI_ERROR_TMO	viMapAddress() could not acquire resource or perform mapping before the timer expired.
VI_ERROR_INV_SIZE	Invalid size of window specified.
VI_ERROR_ALLOC	Unable to allocate window of at least the requested size.
VI_ERROR_INV_ACC_MODE	Invalid access mode.

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).

Description

The `viMapAddress()` 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. The following table lists the valid entries for the **mapSpace** parameter.

Value	Description
VXI and VME	VI_A16_SPACE (1) VI_A24_SPACE (2) VI_A32_SPACE (3) VI_A64_SPACE (4)
PXI INSTR	VI_PXI_CFG_SPACE (10) VI_PXI_BAR0_SPACE (11) to VI_PXI_BAR5_SPACE (16)
PXI MEMACC	VI_PXI_ALLOC_SPACE (9)



**Note** On some hardware platforms, the low-level driver may have limitations on the parameters to this function. For example, on VXI resources **mapBase** should be a multiple of **mapSize** for best results. If these limitations prevent NI-VISA from mapping the full region you request (**mapSize** bytes starting at **mapBase**), the function will return an error such as `VI_ERROR_NSUP_OFFSET` or `VI_ERROR_ALLOC`.

## INSTR Specific

Notice that **mapBase** specified in the `viMapAddress()` operation for an INSTR Resource is the offset address relative to the device's allocated address base for the corresponding address space that was specified. For example, if **mapSpace** specifies `VI_A16_SPACE`, then **mapBase** specifies the offset from the logical address base address of the specified VXI device. If **mapSpace** specifies `VI_A24_SPACE` or `VI_A32_SPACE`, then **mapBase** specifies the offset from the base address of the VXI device's memory space allocated by the VXI Resource Manager within VXI A24 or A32 space.

## MEMACC Specific

For a MEMACC Resource, the **mapBase** parameter specifies an absolute address.



**Note** The output address is not necessarily always a pointer. It may be possible for `viMapAddress` to succeed and output a token address value of 0. This is not the same as a NULL pointer, even though the value of NULL is 0. Obviously, this situation cannot happen if the address is a pointer that the user can dereference. Regardless, you should determine whether `viMapAddress` succeeded or failed by checking the returned status, not the output value of the address.

## Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[viUnmapAddress](#)

## viMapTrigger

### Purpose

Map the specified trigger source line to the specified destination line.

## C Syntax

```
viStatus viMapTrigger(ViSession vi, ViInt16 trigSrc, ViInt16 trigDest, ViUInt16 mode)
```

## Visual Basic Syntax

```
viMapTrigger& (ByVal vi&, ByVal trigSrc%, ByVal trigDest%, ByVal mode%)
```

## Resource Classes

PXI BACKPLANE, VXI BACKPLANE

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>trigSrc</b>	IN	Source line from which to map. Refer to the <b>Description</b> section for actual values.
<b>trigDest</b>	IN	Destination line to which to map. Refer to the <b>Description</b> section for actual values.
<b>mode</b>	IN	VI_NULL

## Return Values

Completion Codes	Description
------------------	-------------

VI_SUCCESS	Operation completed successfully.
VI_SUCCESS_TRIG_MAPPED	The path from <b>trigSrc</b> to <b>trigDest</b> is already mapped.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given <b>vi</b> does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by <b>vi</b> has been locked for this kind of access.
VI_ERROR_TMO	Timeout expired before operation completed.
VI_ERROR_INV_MODE	The value specified by the <b>mode</b> parameter is invalid.
VI_ERROR_LINE_IN_USE	One of the specified lines ( <b>trigSrc</b> or <b>trigDest</b> ) is currently in use.
VI_ERROR_INV_LINE	One of the specified lines ( <b>trigSrc</b> or <b>trigDest</b> ) is invalid.
VI_ERROR_NSUP_LINE	One of the specified lines ( <b>trigSrc</b> or <b>trigDest</b> ) is not supported by this VISA implementation.



## Description

This operation can be used to map one trigger line to another. This operation is valid only on BACKPLANE (mainframe) sessions.

Value	Action Description
VI_TRIG_TTL0 - VI_TRIG_TTL7	Map the specified VXI or PXI TTL trigger line.
VI_TRIG_ECL0 - VI_TRIG_ECL1	Map the specified VXI ECL trigger line.
VI_TRIG_PANEL_IN	Map the controller's front panel trigger input line.
VI_TRIG_PANEL_OUT	Map the controller's front panel trigger output line.

If this operation is called multiple times on the same BACKPLANE Resource with the same source trigger line and different destination trigger lines, the result will be that when the source trigger line is asserted, all of the specified destination trigger lines will also be asserted. If this operation is called multiple times on the same BACKPLANE Resource with different source trigger lines and the same destination trigger line, the result will be that when any of the specified source trigger lines is asserted, the destination trigger line will also be asserted.



**Note** Mapping a trigger line (as either source or destination) multiple times requires special hardware capabilities and is not guaranteed to be implemented.

Refer to [VI\\_ATTR\\_PXI\\_SRC\\_TRIG\\_BUS](#) or [VI\\_ATTR\\_PXI\\_DEST\\_TRIG\\_BUS](#) for information about how to map a trigger between bus segments in a multisegment PXI chassis.

---

## Related Topics

BACKPLANE Resource

VI\_ATTR\_PXI\_DEST\_TRIG\_BUS

VI\_ATTR\_PXI\_SRC\_TRIG\_BUS

viMemAlloc/viMemAllocEx

Purpose

Allocates memory from a resource's memory region.

C Syntax

```
ViStatus viMemAlloc(ViSession vi, ViBusSize size, ViPBusAddress offset)
```

```
ViStatus viMemAllocEx(ViSession vi, ViBusSize size, ViPBusAddress64 offset)
```

Visual Basic Syntax

```
viMemAlloc&(ByVal vi&, ByVal size&, offset&)
```

Resource Classes

PXI MEMACC, VXI INSTR

Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.

<b>size</b>	IN	Specifies the size of the allocation.
<b>offset</b>	OUT	Returns the offset of the allocated memory. For <code>viMemAlloc()</code> , this is a 32-bit value for 32-bit applications and a 64-bit value for 64-bit applications. For <code>viMemAllocEx()</code> , this is always a 64-bit value.

## Return Values

Completion Codes	Description
<code>VI_SUCCESS</code>	Operation completed successfully.

Error Codes	Description
<code>VI_ERROR_INV_OBJECT</code>	The given session reference is invalid.
<code>VI_ERROR_NSUP_OPER</code>	The given <b>vi</b> does not support this operation.
<code>VI_ERROR_RSRC_LOCKED</code>	Specified operation could not be performed because the resource identified by <b>vi</b> has been locked for this kind of access.
<code>VI_ERROR_INV_SIZE</code>	Invalid <b>size</b> specified.
<code>VI_ERROR_ALLOC</code>	Unable to allocate shared memory block of the requested <b>size</b> .

<code>VI_ERROR_MEM_NSHARED</code>	The device does not export any memory.
-----------------------------------	--

## Description

The memory region referenced by the **offset** returned from `viMemAlloc()` can be accessed with the high-level operations `viMoveInXX()` and `viMoveOutXX()`, or mapped using `viMapAddress()`. When using `viMemAllocEx()`, the **offset** returned may be accessed by the `viMoveInXXEx()` and `viMoveOutXXEx()` operations, and mapped using `viMapAddressEx()`. Note that for `viMemAllocEx()`, the offset could be above the 4 GB boundary. If your device cannot access this memory, you should use `viMemAlloc()` instead.

### VXI INSTR Specific

Notice that the **offset** parameter to these operations for an INSTR Resource is the offset address relative to the device's allocated address base. The `viMemAlloc()` and `viMemAllocEx()` operations return an **offset** into a device's memory region allocated for use by this session. If the device to which the given **vi** refers is on the local interface card, the memory can be allocated either on the device itself or on the computer's system memory.

### PXI/PCI MEMACC Specific

For a MEMACC Resource, the **offset** parameter specifies an absolute address. This is a physical address in system memory and can be used for device DMA.

---

## Related Topics

[INSTR Resource](#)

[viMapAddress/viMapAddressEx](#)

[viMemFree/viMemFreeEx](#)

[viMoveIn8/viMoveIn16/viMoveIn32/viMoveIn64, viMoveIn8Ex/viMoveIn16Ex/](#)

[viMoveIn32Ex/viMoveIn64Ex](#)

[viMoveOut8/viMoveOut16/viMoveOut32/viMoveOut64, viMoveOut8Ex/viMoveOut16Ex/viMoveOut32Ex/viMoveOut64Ex](#)

# viMemFree/viMemFreeEx

## Purpose

Frees memory previously allocated using the `viMemAlloc()` operation.

## C Syntax

```
ViStatus viMemFree(ViSession vi, ViBusAddress offset)
```

```
ViStatus viMemFreeEx(ViSession vi, ViBusAddress64 offset)
```

## Visual Basic Syntax

```
viMemFree&(ByVal vi&, ByVal offset&)
```

## Resource Classes

PXI MEMACC, VXI INSTR

## Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
offset	IN	Specifies the memory previously allocated with <code>viMemAlloc()</code> or <code>viMemAllocEx</code> . For <code>viMemFree()</code> , this is a 32-bit value for 32-bit applications and a 64-bit value for 64-bit applications. For <code>viMemFreeEx()</code> , this is always a 64-bit value.

## Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_INV_OFFSET	Invalid <b>offset</b> specified.
VI_ERROR_WINDOW_MAPPED	The specified <b>offset</b> is currently in use by <code>viMapAddress()</code> .

## Description

The `viMemFree()` operation frees the memory previously allocated using `viMemAlloc()`. The `viMemFreeEx()` operation frees the memory previously allocated using `viMemAllocEx()`. If the specified offset has been mapped using `viMapAddress()` or `viMapAddressEx()`, it must be unmapped before it can be freed.

## Related Topics

[INSTR Resource](#)

[viMapAddress/viMapAddressEx](#)

[viMemAlloc/viMemAllocEx](#)

[viUnmapAddress](#)

# viMove/viMoveEx

## Purpose

Moves a block of data.

## C Syntax

```
ViStatus viMove(ViSession vi, ViUInt16 srcSpace, ViBusAddress
srcOffset, ViUInt16 srcWidth, ViUInt16 destSpace, ViBusAddress des
tOffset, ViUInt16 destWidth, ViBusSize length)
```

```
ViStatus viMoveEx(ViSession vi, ViUInt16 srcSpace, ViBusAddres
s64 srcOffset, ViUInt16 srcWidth, ViUInt16 destSpace, ViBusAddres
s64 destOffset, ViUInt16 destWidth, ViBusSize length)
```

## Visual Basic Syntax

```
viMove&(ByVal vi&, ByVal srcSpace%, ByVal srcOffset&, ByVal srcWid
th%, ByVal destSpace%, ByVal destOffset&, ByVal destWidth%, ByVal
length&)
```

## Resource Classes

VXI INSTR, VXI MEMACC

## Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.

<b>srcSpace</b>	IN	Specifies the address space of the source.
<b>srcOffset</b>	IN	Offset of the starting address or register from which to read. For <code>viMove()</code> , this is a 32-bit value for 32-bit applications and a 64-bit value for 64-bit applications. For <code>viMoveEx()</code> , this is always a 64-bit value.
<b>srcWidth</b>	IN	Specifies the data width of the source.
<b>destSpace</b>	IN	Specifies the address space of the destination.
<b>destOffset</b>	IN	Offset of the starting address or register to which to write. For <code>viMove()</code> , this is a 32-bit value for 32-bit applications and a 64-bit value for 64-bit applications. For <code>viMoveEx()</code> , this is always a 64-bit value.
<b>destWidth</b>	IN	Specifies the data width of the destination.
<b>length</b>	IN	Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.

## Return Values

Completion Codes	Description
<code>VI_SUCCESS</code>	Operation completed successfully.

Error Codes	Description
-------------	-------------



VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_INV_SPACE	Invalid source or destination space specified.
VI_ERROR_INV_OFFSET	Invalid source or destination offset specified.
VI_ERROR_INV_WIDTH	Invalid source or destination width specified.
VI_ERROR_NSUP_OFFSET	Specified source or destination offset is not accessible from this hardware.
VI_ERROR_NSUP_VAR_WIDTH	Cannot support source and destination widths that are different.
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_NSUP_WIDTH	Specified width is not supported by this hardware.

<code>VI_ERROR_NSUP_ALIGN_OFFSET</code>	The specified offset is not properly aligned for the access width of the operation.
<code>VI_ERROR_INV_LENGTH</code>	Invalid length specified.

## Description

The `viMove()` and `viMoveEx()` operations move data from the specified source to the specified destination. The source and the destination can either be local memory or the offset of the interface with which this MEMACC Resource is associated. These operations use the specified data width and address space. In some systems, such as VXI, users can specify additional settings for the transfer, such as byte order and access privilege, by manipulating the appropriate attributes.

The following table lists the valid entries for specifying address space.

Value	Description
<code>VI_A16_SPACE (1)</code>	Address the A16 address space of the VXI/MXI bus.
<code>VI_A24_SPACE (2)</code>	Address the A24 address space of the VXI/MXI bus.
<code>VI_A32_SPACE (3)</code>	Address the A32 address space of the VXI/MXI bus.
<code>VI_LOCAL_SPACE (0)</code>	Address process-local memory (using a virtual address).
<code>VI_OPAQUE_SPACE (FFFh)</code>	Addresses potentially volatile data (using a virtual address).

The following table lists the valid entries for specifying widths.

Value	Description
<code>VI_WIDTH_8 (1)</code>	Performs 8-bit (D08) transfers.
<code>VI_WIDTH_16 (2)</code>	Performs 16-bit (D16) transfers.
<code>VI_WIDTH_32 (4)</code>	Performs 32-bit (D32) transfers.
<code>VI_WIDTH_64 (8)</code>	Performs 64-bit (D64) transfers.

All VXI accesses performed by the `viMove()` and `viMoveEx()` operations use either the same or successive offsets, depending on the increment value specified by `VI_ATTR_SRC_INCREMENT` and `VI_ATTR_DEST_INCREMENT`.

If **srcSpace** is `VI_LOCAL_SPACE`, `viMove()` will ignore `VI_ATTR_SRC_INCREMENT`. If **destSpace** is `VI_LOCAL_SPACE`, `viMove()` will ignore `VI_ATTR_DEST_INCREMENT`. Local accesses always increment the offset for each index in a multi-element transfer, rather than using the increment specified by the attributes. If **srcSpace** is any value other than `VI_LOCAL_SPACE`, including `VI_OPAQUE_SPACE`, `viMove()` will honor `VI_ATTR_SRC_INCREMENT`. If **destSpace** is any value other than `VI_LOCAL_SPACE`, including `VI_OPAQUE_SPACE`, `viMove()` will honor `VI_ATTR_DEST_INCREMENT`. While `VI_OPAQUE_SPACE` uses a process-local virtual address, it is not necessarily pointing to system memory, so it may be a FIFO. Therefore, `VI_ATTR_SRC/DEST_INCREMENT` do indeed apply.

## INSTR Specific

If **srcSpace** is neither `VI_LOCAL_SPACE` nor `VI_OPAQUE_SPACE`, **srcOffset** is a relative address of the device associated with the given INSTR resource. Similarly, if **destspace** is neither `VI_LOCAL_SPACE` nor `VI_OPAQUE_SPACE`, **destOffset** is a relative address of the device associated with the given INSTR resource.

The primary intended use of this operation with an INSTR session is to synchronously

move data to or from the device. Therefore, either the **srcSpace** or **destSpace** parameter will usually be `VI_LOCAL_SPACE`.

## MEMACC Specific

The **destOffset** and **srcOffset** parameters specify absolute addresses. Notice also that the **length** specified in the `viMove()` and `viMoveEx()` operations is the number of elements (of the size corresponding to the **srcWidth** parameter) to transfer, beginning at the specified offsets. Therefore, **srcOffset + length\*srcWidth** cannot exceed the total amount of memory exported by the given **srcSpace**. Similarly, **destOffset + length\*srcWidth** cannot exceed the total amount of memory exported by the given **destSpace**.

---

### Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[VI\\_ATTR\\_DEST\\_INCREMENT](#)

[VI\\_ATTR\\_SRC\\_INCREMENT](#)

[viMoveAsync/viMoveAsyncEx](#)

## viMoveAsync/viMoveAsyncEx

### Purpose

Moves a block of data asynchronously.

### C Syntax

```
ViStatus viMoveAsync(ViSession vi, ViUInt16 srcSpace, ViBusAddress srcOffset, ViUInt16 srcWidth, ViUInt16 destSpace, ViBusAddress
```

```
ss destOffset, ViUInt16 destWidth, ViBusSize length, ViPJobId jobId)
```

```
ViStatus viMoveAsyncEx(ViSession vi, ViUInt16 srcSpace, ViBusAddress64 srcOffset, ViUInt16 srcWidth, ViUInt16 destSpace, ViBusAddress64 destOffset, ViUInt16 destWidth, ViBusSize length, ViPJobId jobId)
```

## Visual Basic Syntax

N/A

## Resource Classes

PXI INSTR, VXI INSTR, VXI MEMACC

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>srcSpace</b>	IN	Specifies the address space of the source.
<b>srcOffset</b>	IN	Offset of the starting address or register from which to read. For <code>viMove()</code> , this is a 32-bit value for 32-bit applications and a 64-bit value for 64-bit applications. For <code>viMoveEx()</code> , this is always a 64-bit value.
<b>srcWidth</b>	IN	Specifies the data width of the source.
<b>destSpace</b>	IN	Specifies the address space of the destination.

<b>destOffset</b>	IN	Offset of the starting address or register to which to write. For <code>viMove()</code> , this is a 32-bit value for 32-bit applications and a 64-bit value for 64-bit applications. For <code>viMoveEx()</code> , this is always a 64-bit value.
<b>destWidth</b>	IN	Specifies the data width of the destination.
<b>length</b>	IN	Number of elements to transfer, where the data width of the elements to transfer is identical to the source data width.
<b>jobId</b>	OUT	Job identifier of this asynchronous move operation.

## Return Values

Completion Codes	Description
<code>VI_SUCCESS</code>	Asynchronous operation successfully queued.
<code>VI_SUCCESS_SYNC</code>	Operation performed synchronously.

Error Codes	Description
<code>VI_ERROR_INV_OBJECT</code>	The given session reference is invalid.
<code>VI_ERROR_NSUP_OPER</code>	The given <code>vi</code> does not support this operation.
<code>VI_ERROR_RSRC_LOCKED</code>	Specified operation could not be performed because the

	resource identified by <code>vi</code> has been locked for this kind of access.
<code>VI_ERROR_QUEUE_ERROR</code>	Unable to queue move operation (usually due to the I/O completion event not being enabled or insufficient space in the session's queue).
<code>VI_ERROR_IN_PROGRESS</code>	Unable to queue the asynchronous operation because there is already an operation in progress.

## Description

The `viMoveAsync()` and `viMoveAsyncEx()` operations asynchronously move data from the specified source to the specified destination. This operation queues up the transfer in the system, then it returns immediately without waiting for the transfer to carry out or complete. When the transfer terminates, a `VI_EVENT_IO_COMPLETION` event is generated, which indicates the status of the transfer.

This operation returns **jobId**, which you can use either with `viTerminate()` to abort the operation or with `VI_EVENT_IO_COMPLETION` events to identify which asynchronous move operations completed. VISA will never return `VI_NULL` for a valid **jobId**.

The source and the destination can either be local memory or the offset of the interface with which this INSTR or MEMACC Resource is associated. This operation uses the specified data width and address space. In some systems, such as VXI, users can specify additional settings for the transfer, such as byte order and access privilege, by manipulating the appropriate attributes.

The following table lists the valid entries for specifying address space.

Value	Description
-------	-------------

<code>VI_A16_SPACE (1)</code>	Address the A16 address space of the VXI/MXI bus.
<code>VI_A24_SPACE (2)</code>	Address the A24 address space of the VXI/MXI bus.
<code>VI_A32_SPACE (3)</code>	Address the A32 address space of the VXI/MXI bus.
<code>VI_LOCAL_SPACE (0)</code>	Address process-local memory (using a virtual address).
<code>VI_OPAQUE_SPACE (FFFh)</code>	Addresses potentially volatile data (using a virtual address).

The following table lists the valid entries for specifying widths.

Value	Description
<code>VI_WIDTH_8 (1)</code>	Performs 8-bit (D08) transfers.
<code>VI_WIDTH_16 (2)</code>	Performs 16-bit (D16) transfers.
<code>VI_WIDTH_32 (4)</code>	Performs 32-bit (D32) transfers.
<code>VI_WIDTH_64 (8)</code>	Performs 64-bit (D64) transfers.

All VXI and PXI accesses performed by the `viMoveAsync()` and `viMoveAsyncEx()` operations use either the same or successive offsets, depending on the increment value specified by `VI_ATTR_SRC_INCREMENT` and `VI_ATTR_DEST_INCREMENT`.

If `srcSpace` is `VI_LOCAL_SPACE`, `viMove()` will ignore `VI_ATTR_SRC_INCREMENT`



NT. If **destSpace** is `VI_LOCAL_SPACE`, `viMove()` will ignore `VI_ATTR_DEST_INCREMENT`. Local accesses always increment the offset for each index in a multi-element transfer, rather than using the increment specified by the attributes. If **srcSpace** is any value other than `VI_LOCAL_SPACE`, including `VI_OPAQUE_SPACE`, `viMove()` will honor `VI_ATTR_SRC_INCREMENT`. If **destSpace** is any value other than `VI_LOCAL_SPACE`, including `VI_OPAQUE_SPACE`, `viMove()` will honor `VI_ATTR_DEST_INCREMENT`. While `VI_OPAQUE_SPACE` uses a process-local virtual address, it is not necessarily pointing to system memory, so it may be a FIFO. Therefore, `VI_ATTR_SRC/DEST_INCREMENT` do indeed apply.

## INSTR Specific

If **srcSpace** is neither `VI_LOCAL_SPACE` nor `VI_OPAQUE_SPACE`, **srcOffset** is a relative address of the device associated with the given INSTR resource. Similarly, if **destspace** is neither `VI_LOCAL_SPACE` nor `VI_OPAQUE_SPACE`, **destOffset** is a relative address of the device associated with the given INSTR resource.

The primary intended use of this operation with an INSTR session is to asynchronously move data to or from the device. Therefore, either the **srcSpace** or **destSpace** parameter will usually be `VI_LOCAL_SPACE`.

## MEMACC Specific

The **destOffset** and **srcOffset** parameters specify absolute addresses. Notice also that the length specified in the `viMoveAsync()` and `viMoveAsyncEx()` operations is the number of elements (of the size corresponding to the **srcWidth** parameter) to transfer, beginning at the specified offsets. Therefore, **srcOffset + length\*srcWidth** cannot exceed the total amount of memory exported by the given **srcSpace**. Similarly, **destOffset + length\*srcWidth** cannot exceed the total amount of memory exported by the given **destSpace**.

---

## Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

VI\_ATTR\_DEST\_INCREMENTVI\_ATTR\_SRC\_INCREMENTVI\_EVENT\_IO\_COMPLETIONviMove/viMoveEx

viMoveIn8/viMoveIn16/viMoveIn32/viMoveIn64,  
viMoveIn8Ex/viMoveIn16Ex/viMoveIn32Ex/viMoveIn64Ex

## Purpose

Moves a block of data from the specified address space and offset to local memory.

## C 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)
```

```
ViStatus viMoveIn64(ViSession vi, ViUInt16 space, ViBusAddress
offset, ViBusSize length, ViAUInt64 buf64)
```

```
ViStatus viMoveIn8Ex(ViSession vi, ViUInt16 space, ViBusAddress
offset, ViBusSize length, ViAUInt8 buf8)
```

```
ViStatus viMoveIn16Ex(ViSession vi, ViUInt16 space, ViBusAddress
offset, ViBusSize length, ViAUInt16 buf16)
```

```
ViStatus viMoveIn32Ex(ViSession vi, ViUInt16 space, ViBusAddress
```

```
ess64 offset, ViBusSize length, ViAUInt32 buf32)
```

```
ViStatus viMoveIn64Ex(ViSession vi, ViUInt16 space, ViBusAddress64 offset, ViBusSize length, ViAUInt64 buf64)
```

## Visual Basic Syntax

```
viMoveIn8&(ByVal vi&, ByVal space%, ByVal offset&, ByVal length&, buf8 as Byte)
```

```
viMoveIn16&(ByVal vi&, ByVal space%, ByVal offset&, ByVal length&, buf16%)
```


```
viMoveIn32&(ByVal vi&, ByVal space%, ByVal offset&, ByVal length&, buf32&)
```

## Resource Classes

PXI INSTR, PXI MEMACC, VXI INSTR, VXI MEMACC

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>space</b>	IN	Specifies the address space. Refer to the table included in the <b>Description</b> section.
<b>offset</b>	IN	Offset (in bytes) of the starting address to read. For <b>viMoveInX X()</b> operations, this is a 32-bit value for 32-bit applications and a 64-bit value for 64-bit applications. For <b>viMoveInXXEx()</b> operations, this is always a 64-bit value.

		<div>  <p><b>Note</b> VISA Out and VISA In functions require the offset to begin at a value evenly divisible by the number of bytes being accessed. For example, VISA Out/In 32 requires an offset evenly divisible by 4 bytes, so valid offset values could be 0x00, 0x04, 0x08, 0x0C, 0x10, 0x14, and so on. Values other than these return an error saying the offset is not properly aligned.</p> </div>
<b>length</b>	IN	Number of elements to transfer, where the data width of the elements to transfer is identical to data width (8, 16, 32, or 64 bits).
<b>buf8, buf16, buf32, or buf64</b>	OUT	Data read from bus (8 bits for <code>viMoveIn8 [Ex] ()</code> , 16 bits for <code>viMoveIn16 [Ex] ()</code> , 32 bits for <code>viMoveIn32 [Ex] ()</code> , and 64 bits for <code>viMoveIn64 [Ex] ()</code> ).

## Return Values

Completion Codes	Description
<code>VI_SUCCESS</code>	Operation completed successfully.

Error Codes	Description
<code>VI_ERROR_INV_OBJECT</code>	The given session reference is invalid.
<code>VI_ERROR_NSUP_OPER</code>	The given vi does not support this operation.
<code>VI_ERROR_RSRC_LOCKED</code>	Specified operation could not be performed because the

	resource identified by <code>vi</code> has been locked for this kind of access.
<code>VI_ERROR_BERR</code>	Bus error occurred during transfer.
<code>VI_ERROR_INV_SPACE</code>	Invalid address space specified.
<code>VI_ERROR_INV_OFFSET</code>	Invalid offset specified.
<code>VI_ERROR_NSUP_OFFSET</code>	Specified offset is not accessible from this hardware.
<code>VI_ERROR_NSUP_WIDTH</code>	Specified width is not supported by this hardware.
<code>VI_ERROR_INV_LENGTH</code>	Invalid length specified.
<code>VI_ERROR_NSUP_ALIGN_OFFSET</code>	The specified offset is not properly aligned for the access width of the operation.
<code>VI_ERROR_INV_SETUP</code>	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).

## Description

The `viMoveInXX[Ex]()` operations use the specified address space to read in 8, 16, 32, or 64 bits of data, respectively, from the specified offset. These operations do not require `viMapAddress()` to be called prior to their invocation.

The following table lists the valid entries for specifying address space.

Value	Description
VXI and VME	VI_A16_SPACE (1) VI_A24_SPACE (2) VI_A32_SPACE (3) VI_A64_SPACE (4)
PXI INSTR	VI_PXI_CFG_SPACE (10) VI_PXI_BAR0_SPACE (11) to VI_PXI_BAR5_SPACE (16)
PXI MEMACC	VI_PXI_ALLOC_SPACE (9)

For these operations, VISA ignores the attribute `VI_ATTR_DEST_INCREMENT` and increments the local buffer address for each element. It is valid for the VISA driver to copy the data into the user buffer at any width it wants. In other words, even if the width is a byte (8-bit), the VISA driver can perform 32-bit PCI burst accesses because it is just memory, to improve throughput.

## INSTR Specific

Notice that the **offset** parameter to these operations for an INSTR Resource is the offset address relative to the device's allocated address base for the corresponding address space that was specified. For example, if **space** specifies `VI_A16_SPACE`, then **offset** specifies the offset from the logical address base address of the specified VXI device. If **space** specifies `VI_A24_SPACE` or `VI_A32_SPACE`, then **offset** specifies the offset from the base address of the VXI device's memory space allocated by the VXI Resource Manager within VXI A24 or A32 space.

Notice also that the **length** specified in the `viMoveInXX()` operations for an INSTR Resource is the number of elements (of the **size** corresponding to the operation) to transfer, beginning at the specified **offset**. Therefore, **offset + length\*size** cannot exceed the amount of memory exported by the device in the given **space**.

## MEMACC Specific

For a MEMACC Resource, the **offset** parameter specifies an absolute address.

Notice also that the **length** parameter to these operations for a MEMACC Resource is the number of elements (of the **size** corresponding to the operation) to transfer, beginning at the specified **offset**. Therefore, **offset + length\*size** cannot exceed the total amount of memory available in the given **space**.

### Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[VI\\_ATTR\\_DEST\\_INCREMENT](#)

[viMoveOut8/viMoveOut16/viMoveOut32/viMoveOut64, viMoveOut8Ex/viMoveOut16Ex/viMoveOut32Ex/viMoveOut64Ex](#)

viMoveOut8/viMoveOut16/viMoveOut32/viMoveOut64,  
viMoveOut8Ex/viMoveOut16Ex/viMoveOut32Ex/  
viMoveOut64Ex

### Purpose

Moves a block of data from local memory to the specified address space and offset.

### C Syntax

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

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

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

```
ViStatus viMoveOut64(ViSession vi, ViUInt16 space, ViBusAddress offset, ViBusSize length, ViAUInt64 buf64)
```

```
ViStatus viMoveOut8Ex(ViSession vi, ViUInt16 space, ViBusAddress64 offset, ViBusSize length, ViAUInt8 buf8)
```

```
ViStatus viMoveOut16Ex(ViSession vi, ViUInt16 space, ViBusAddress64 offset, ViBusSize length, ViAUInt16 buf16)
```

```
ViStatus viMoveOut32Ex(ViSession vi, ViUInt16 space, ViBusAddress64 offset, ViBusSize length, ViAUInt32 buf32)
```

```
ViStatus viMoveOut64Ex(ViSession vi, ViUInt16 space, ViBusAddress64 offset, ViBusSize length, ViAUInt64 buf64)
```

## Visual Basic Syntax

```
viMoveOut8&(ByVal vi&, ByVal space%, ByVal offset&, ByVal lengthh&, buf8 as Byte)
```

```
viMoveOut16&(ByVal vi&, ByVal space%, ByVal offset&, ByVal lengthh&, buf16%)
```

```
viMoveOut32&(ByVal vi&, ByVal space%, ByVal offset&, ByVal lengthh&, buf32&)
```


## Resource Classes

PXI INSTR, PXI MEMACC, VXI INSTR, VXI MEMACC

## Parameters

Name	Direction	Description
------	-----------	-------------



<b>vi</b>	IN	Unique logical identifier to a session.
<b>space</b>	IN	Specifies the address space. Refer to the table included in the <b>Description</b> section.
<b>offset</b>	IN	<p>Offset (in bytes) of the device to write to. For <code>viMoveOutXX()</code> operations, this is a 32-bit value for 32-bit applications and a 64-bit value for 64-bit applications. For <code>viMoveOutXXEx()</code> operations, this is always a 64-bit value.</p> <div>  <p><b>Note</b> VISA Out and VISA In functions require the offset to begin at a value evenly divisible by the number of bytes being accessed. For example, VISA Out/In 32 requires an offset evenly divisible by 4 bytes, so valid offset values could be 0x00, 0x04, 0x08, 0x0C, 0x10, 0x14, and so on. Values other than these return an error saying the offset is not properly aligned.</p> </div>
<b>length</b>	IN	Number of elements to transfer, where the data width of the elements to transfer is identical to data width (8, 16, 32, or 64 bits).
<b>buf8, buf16, buf32, or buf64</b>	IN	Data to write to bus (8 bits for <code>viMoveOut8[Ex]()</code> , 16 bits for <code>viMoveOut16[Ex]()</code> , 32 bits for <code>viMoveOut32[Ex]()</code> , and 64 bits for <code>viMoveOut64[Ex]()</code> ).

## Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
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.
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to

	attributes being set to an inconsistent state).
--	---

Description

The `viMoveOutXX[Ex] ()` operations use the specified address space to write 8, 16, 32, or 64 bits of data, respectively, to the specified offset. These operations do not require `viMapAddress ()` to be called prior to their invocation.

The following table lists the valid entries for specifying address space.

Interface	Values
VXI and VME	<code>VI_A16_SPACE (1)</code> <code>VI_A24_SPACE (2)</code> <code>VI_A32_SPACE (3)</code> <code>VI_A64_SPACE (4)</code>
PXI INSTR	<code>VI_PXI_CFG_SPACE (10)</code> <code>VI_PXI_BAR0_SPACE (11) to VI_PXI_BAR5_SPACE (16)</code>
PXI MEMACC	<code>VI_PXI_ALLOC_SPACE (9)</code>

For these operations, VISA ignores the attribute `VI_ATTR_SRC_INCREMENT` and increments the local buffer address for each element. It is valid for the VISA driver to copy the data out of the user buffer at any width it wants. In other words, even if the width is a byte (8-bit), the VISA driver can perform 32-bit PCI burst accesses because it is just memory, to improve throughput.

INSTR Specific

Notice that the **offset** parameter to these operations for an INSTR Resource is the offset address relative to the device's allocated address base for the corresponding address space that was specified. For example, if **space** specifies `VI_A16_SPACE`,

then **offset** specifies the offset from the logical address base address of the specified VXI device. If **space** specifies `VI_A24_SPACE` or `VI_A32_SPACE`, then **offset** specifies the offset from the base address of the VXI device's memory space allocated by the VXI Resource Manager within VXI A24 or A32 space.

Notice also that the **length** specified in the `viMoveInXX()` operations for an INSTR Resource is the number of elements (of the **size** corresponding to the operation) to transfer, beginning at the specified **offset**. Therefore, **offset + length\*size** cannot exceed the amount of memory exported by the device in the given **space**.

## MEMACC Specific

For a MEMACC Resource, the **offset** parameter specifies an absolute address.

Notice also that the **length** parameter to these operations for a MEMACC Resource is the number of elements (of the **size** corresponding to the operation) to transfer, beginning at the specified **offset**. Therefore, **offset + length\*size** cannot exceed the total amount of memory available in the given **space**.

---

## Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[VI\\_ATTR\\_DEST\\_INCREMENT](#)

[viMoveIn8/viMoveIn16/viMoveIn32/viMoveIn64, viMoveIn8Ex/viMoveIn16Ex/viMoveIn32Ex/viMoveIn64Ex](#)

## viOpen

### Purpose

Opens a session to the specified resource.

## C Syntax

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

## Visual Basic Syntax

```
viOpen&(ByVal sesn&, ByVal rsrcName$, ByVal accessMode&, ByVal
openTimeout&, vi&)
```

## Resource Classes

### VISA Resource Manager

## Parameters

Name	Direction	Description
<b>sesn</b>	IN	Resource Manager session (should always be a session returned from <code>viOpenDefaultRM()</code> ).
<b>rsrcName</b>	IN	Unique symbolic name of a resource. Refer to the <b>Description</b> section for more information.
<b>accessMode</b>	IN	Specifies the mode by which the resource is to be accessed. Refer to the <b>Description</b> section for valid values. If the parameter value is <code>VI_NULL</code> , the session uses VISA-supplied default values.
<b>openTimeout</b>	IN	Specifies the maximum time period (in milliseconds) that this operation waits before returning an error. This does not set the I/O timeout – to do that you must call <code>viSetAttribute()</code> with the attribute <code>VI_ATTR_TMO_VALUE</code> .

<b>vi</b>	OUT	Unique logical identifier reference to a session.
-----------	-----	---

## Return Values

Completion Codes	Description
VI_SUCCESS	Session opened successfully.
VI_SUCCESS_DEV_NPRESENT	Session opened successfully, but the device at the specified address is not responding.
VI_WARN_CONFIG_NLOADED	The specified configuration either does not exist or could not be loaded; using VISA-specified defaults.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given <b>sesn</b> does not support this operation. This operation is supported only by a Resource Manager session.
VI_ERROR_INV_RSRC_NAME	Invalid resource reference specified. Parsing error.
VI_ERROR_INV_ACC_MODE	Invalid access mode.
VI_ERROR_RSRC_NFOUND	Insufficient location information or resource not present

	in the system.
VI_ERROR_ALLOC	Insufficient system resources to open a session.
VI_ERROR_RSRC_BUSY	The resource is valid, but VISA cannot currently access it.
VI_ERROR_RSRC_LOCKED	Specified type of lock cannot be obtained because the resource is already locked with a lock type incompatible with the lock requested.
VI_ERROR_TMO	A session to the resource could not be obtained within the specified openTimeout period.
VI_ERROR_LIBRARY_NFOUND	A code library required by VISA could not be located or loaded.
VI_ERROR_INTF_NUM_NCONFIG	The interface type is valid, but the specified interface number is not configured.
VI_ERROR_MACHINE_NAVAIL	The remote machine does not exist or is not accepting any connections. If the NI-VISA server is installed and running on the remote machine, it may have an incompatible version or may be listening on a different port.
VI_ERROR_NPERMISSION	Access to the remote machine is denied.

## Description

The `viOpen()` operation opens a session to the specified resource. It returns a session identifier that can be used to call any other operations of that resource. The address string passed to `viOpen()` must uniquely identify a resource. Refer to [VISA Resource Syntax and Examples](#) for the syntax of resource strings and examples.

For the parameter **accessMode**, the value `VI_EXCLUSIVE_LOCK` (1) is used to acquire an exclusive lock immediately upon opening a session; if a lock cannot be acquired, the session is closed and an error is returned. The value `VI_LOAD_CONFIG` (4) is used to configure attributes to values specified by some external configuration utility. Multiple access modes can be used simultaneously by specifying a **bit-wise OR** of the values other than `VI_NULL`. NI-VISA currently supports `VI_LOAD_CONFIG` only on Serial INSTR sessions.

All resource strings returned by `viFindRsrc()` will always be recognized by `viOpen()`. However, `viFindRsrc()` will not necessarily return all strings that you can pass to `viParseRsrc()` or `viOpen()`. This is especially true for network and TCPIP resources.

---

## Related Topics

[viClose](#)

[viFindRsrc](#)

[viOpenDefaultRM](#)

[viParseRsrc](#)

[VISA Resource Manager](#)

[VISA Resource Template](#)

## viOpenDefaultRM



# Purpose

This function returns a session to the Default Resource Manager resource.

# C Syntax

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

# Visual Basic Syntax

```
viOpenDefaultRM& (sesn&)
```

# Resource Classes

VISA Resource Manager

# Parameters

Name	Direction	Description
<b>sesn</b>	OUT	Unique logical identifier to a Default Resource Manager session.

# Return Values

Completion Codes	Description
VI_SUCCESS	Session to the Default Resource Manager resource created successfully.
VI_WARN_CONFIG_NLOADED	At least one configured Passport module could not be loaded.

Error Codes	Description
<code>VI_ERROR_SYSTEM_ERROR</code>	The VISA system failed to initialize.
<code>VI_ERROR_ALLOC</code>	Insufficient system resources to create a session to the Default Resource Manager resource.
<code>VI_ERROR_INV_SETUP</code>	Some implementation-specific configuration file is corrupt or does not exist.
<code>VI_ERROR_LIBRARY_NFOUND</code>	A code library required by VISA could not be located or loaded.

## Description

The `viOpenDefaultRM()` 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.

When a Resource Manager session is passed to `viClose()`, not only is that session closed, but also all find lists and device sessions (which that Resource Manager session was used to create) are closed.

---

## Related Topics

[viClose](#)

[viFindRsrc](#)

[viOpen](#)

## VISA Resource Manager

### VISA Resource Template

## viOut8/viOut16/viOut32/viOut64, viOut8Ex/viOut16Ex/ viOut32Ex/viOut64Ex

### Purpose

Writes an 8-bit, 16-bit, 32-bit, or 64-bit value to the specified memory space and offset.

### C 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)
```

```
ViStatus viOut64(ViSession vi, ViUInt16 space, ViBusAddress offset, ViUInt64 val64)
```

```
ViStatus viOut8Ex(ViSession vi, ViUInt16 space, ViBusAddress64 offset, ViUInt8 val8)
```

```
ViStatus viOut16Ex(ViSession vi, ViUInt16 space, ViBusAddress64 offset, ViUInt16 val16)
```

```
ViStatus viOut32Ex(ViSession vi, ViUInt16 space, ViBusAddress64 offset, ViUInt32 val32)
```

```
ViStatus viOut64Ex(ViSession vi, ViUInt16 space, ViBusAddress64 offset, ViUInt64 val64)
```

## Visual Basic Syntax

`viOut8&(ByVal vi&, ByVal space%, ByVal offset&, ByVal val8 as Byte)`


`viOut16&(ByVal vi&, ByVal space%, ByVal offset&, ByVal val16%)`

`viOut32&(ByVal vi&, ByVal space%, ByVal offset&, ByVal val32&)`

## Resource Classes

PXI INSTR, PXI MEMACC, VXI INSTR, VXI MEMACC

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>space</b>	IN	Specifies the address space. Refer to the table included in the <b>Description</b> section for more information.
<b>offset</b>	IN	<div>Offset (in bytes) of the address or register to which to write. For <code>viOut<b>XX</b>()</code> operations, this is a 32-bit value for 32-bit applications and a 64-bit value for 64-bit applications. For <code>viOut<b>XX</b>Ex()</code> operations, this is always a 64-bit value.</div> <div><div></div><div><b>Note</b> VISA Out and VISA In functions require the offset to begin at a value evenly divisible by the number of bytes being accessed. For example, VISA Out/In 32 requires an offset evenly divisible by 4 bytes, so valid offset values could be 0x00, 0x04, 0x08, 0x0C, 0x10, 0x14, and so on. Values other than these return an error saying the offset is not properly aligned.</div></div>

<b>val8, val16, val32, or val64</b>	IN	Data to write to bus (8 bits for <code>viOut8 [Ex] ()</code> , 16 bits for <code>viOut16 [Ex] ()</code> , 32 bits for <code>viOut32 [Ex] ()</code> , and 64 bits for <code>viOut64 [Ex] ()</code> ).
-------------------------------------	----	--

## Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
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_NSUP_ALIGN_OFFSET	The specified offset is not properly aligned for the access width of the operation.
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).

## Description

The `viOutXX[Ex]()` operations use the specified address space to write 8, 16, 32, or 64 bits of data, respectively, from the specified **offset**. These operations do not require `viMapAddress()` to be called prior to their invocation.

The following table lists the valid entries for specifying address space.

Value	Description
VXI and VME	VI_A16_SPACE (1) VI_A24_SPACE (2) VI_A32_SPACE (3) VI_A64_SPACE (4)
PXI INSTR	VI_PXI_CFG_SPACE (10) VI_PXI_BAR0_SPACE (11) to VI_PXI_BAR5_SPACE (16)
PXI MEMACC	VI_PXI_ALLOC_SPACE (9)

## INSTR Specific

Notice that the **offset** parameter to these operations for an INSTR Resource is the offset address relative to the device's allocated address base for the corresponding address space that was specified. For example, if **space** specifies `VI_A16_SPACE`, then **offset** specifies the offset from the logical address base address of the specified VXI device. If **space** specifies `VI_A24_SPACE` or `VI_A32_SPACE`, then **offset** specifies the offset from the base address of the VXI device's memory space allocated by the VXI Resource Manager within VXI A24 or A32 space.

## MEMACC Specific

For a MEMACC Resource, the **offset** parameter specifies an absolute address.

---

### Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[viIn8/viIn16/viIn32/viIn64, viIn8Ex/viIn16Ex/viIn32Ex/viIn64Ex](#)

## viParseRsrc

### Purpose

Parse a resource string to get the interface information.

### C Syntax

```
ViStatus viParseRsrc(ViSession sesn, ViRsrc rsrcName, ViPUInt16 intfType, ViPUInt16 intfNum)
```

# Visual Basic Syntax

```
viParseRsrc&(ByVal sesn&, ByVal rsrcName$, intfType%, intfNum%)
```

## Resource Classes

### VISA Resource Manager

#### Parameters

Name	Direction	Description
sesn	IN	Resource Manager session (should always be the Default Resource Manager for VISA returned from <code>viOpenDefaultRM()</code> ).
rsrcName	IN	Unique symbolic name of a resource.
intfType	OUT	Interface type of the given resource string.
intfNum	OUT	Board number of the interface of the given resource string.

#### Return Values

Completion Codes	Description
VI_SUCCESS	Resource string is valid.

Error Codes	Description
-------------	-------------



<code>VI_ERROR_INV_OBJECT</code>	The given session reference is invalid.
<code>VI_ERROR_NSUP_OPER</code>	The given <b>sesn</b> does not support this operation. For VISA, this operation is supported only by the Default Resource Manager session.
<code>VI_ERROR_INV_RSRC_NAME</code>	Invalid resource reference specified. Parsing error.
<code>VI_ERROR_RSRC_NFOUND</code>	Insufficient location information or resource not present in the system.
<code>VI_ERROR_ALLOC</code>	Insufficient system resources to parse the string.
<code>VI_ERROR_LIBRARY_NFOUND</code>	A code library required by VISA could not be located or loaded.
<code>VI_ERROR_INTF_NUM_NCONFIG</code>	The interface type is valid, but the specified interface number is not configured.

## Description

This operation parses a resource string to verify its validity. It should succeed for all strings returned by `viFindRsrc()` and recognized by `viOpen()`. This operation is useful if you want to know what interface a given resource descriptor would use without actually opening a session to it. Refer to [VISA Resource Syntax and Examples](#) for the syntax of resource strings and examples.

The values returned in **intfType** and **intfNum** correspond to the attributes `VI_ATTR_INTF_TYPE` and `VI_ATTR_INTF_NUM`. These values would be the same if a user opened that resource with `viOpen()` and queried the attributes with `viGetAttribute()`.

Calling `viParseRsrc()` with `"VXI::1::INSTR"` will produce the same results as invoking it with `"vxi::1::instr"`.

---

## Related Topics

[VI\\_ATTR\\_INTF\\_NUM](#)

[VI\\_ATTR\\_INTF\\_TYPE](#)

[viFindRsrc](#)

[viOpen](#)

[viParseRsrcEx](#)

[VISA Resource Template](#)

## viParseRsrcEx

### Purpose

Parse a resource string to get extended interface information.

### C Syntax

```
ViStatus viParseRsrcEx (ViSession sesn, ViRsrc rsrcName, ViPU
Int16 intfType, ViPUInt16 intfNum, ViChar rsrcClass[], ViChar expan
dedUnaliasedName[], ViChar aliasIfExists[]);
```

### Visual Basic Syntax

```
viParseRsrcEx& (ByVal sesn&, ByVal rsrcName$, intfType%, intfNu
m%, ByVal rsrcClass$, ByVal expandedUnaliasedName$, ByVal aliasIfExi
sts$)
```

## Resource Classes

### VISA Resource Manager

#### Parameters

Name	Direction	Description
<b>sesn</b>	IN	Resource Manager session (should always be the Default Resource Manager for VISA returned from <code>viOpenDefaultRM()</code> ).
<b>rsrcName</b>	IN	Unique symbolic name of a resource.
<b>intfType</b>	OUT	Interface type of the given resource string.
<b>intfNum</b>	OUT	Board number of the interface of the given resource string.
<b>rsrcClass</b>	OUT	Specifies the resource class (for example, "INSTR") of the given resource string.
<b>expanded UnaliasedName</b>	OUT	This is the expanded version of the given resource string. The format should be similar to the VISA-defined canonical resource name.
<b>aliasIfExists</b>	OUT	Specifies the user-defined alias for the given resource string.

## Return Values

Completion Codes	Description
VI_SUCCESS	Resource string is valid.
VI_WARN_EXT_FUNC_NIMPL	The operation succeeded, but a lower level driver did not implement the extended functionality.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given <b>sesn</b> does not support this operation. For VISA, this operation is supported only by the Default Resource Manager session.
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 parse the string.
VI_ERROR_LIBRARY_NFOUND	A code library required by VISA could not be located or loaded.
VI_ERROR_INTF_NUM_NCONFIG	The interface type is valid, but the specified interface

	number is not configured.
--	---------------------------

## Description

This operation parses a resource string to verify its validity. It should succeed for all strings returned by `viFindRsrc()` and recognized by `viOpen()`. This operation is useful if you want to know what interface a given resource descriptor would use without actually opening a session to it. Refer to [VISA Resource Syntax and Examples](#) for the syntax of resource strings and examples.

The values returned in **intfType** and **intfNum** correspond to the attributes `VI_ATTR_INTF_TYPE` and `VI_ATTR_INTF_NUM`. These values would be the same if a user opened that resource with `viOpen()` and queried the attributes with `viGetAttribute()`.

The value returned in **unaliasedExpandedName** should in most cases be identical to the VISA-defined canonical resource name. However, there may be cases where the canonical name includes information that the driver may not know until the resource has actually been opened. In these cases, the value returned in this parameter must be semantically similar.

The value returned in **aliasIfExists** allows programmatic access to user-defined aliases.

Calling `viParseRsrc()` with `"VXI::1::INSTR"` will produce the same results as invoking it with `"vxi::1::instr"`.

---

## Related Topics

[VI\\_ATTR\\_INTF\\_NUM](#)

[VI\\_ATTR\\_INTF\\_TYPE](#)

[viFindRsrc](#)

[viOpen](#)

[viParseRsrc](#)

[VISA Resource Template](#)

# viPeek8/viPeek16/viPeek32/viPeek64

## Purpose

Reads an 8-bit, 16-bit, 32-bit, or 64-bit value from the specified address.

## C Syntax

```
void viPeek8(ViSession vi, ViAddr addr, ViPUInt8 val8)
void viPeek16(ViSession vi, ViAddr addr, ViPUInt16 val16)
void viPeek32(ViSession vi, ViAddr addr, ViPUInt32 val32)
void viPeek64(ViSession vi, ViAddr addr, ViPUInt64 val64)
```

## Visual Basic Syntax

```
viPeek8(ByVal vi&, ByVal addr&, val8 as Byte)
viPeek16(ByVal vi&, ByVal addr&, val16%)
viPeek32(ByVal vi&, ByVal addr&, val32&)
```

## Resource Classes

PXI INSTR, PXI MEMACC, VXI INSTR, VXI MEMACC

## Parameters

Name	Direction	Description
------	-----------	-------------

<b>vi</b>	IN	Unique logical identifier to a session.
<b>addr</b>	IN	Source address to read the value.
<b>val8, val16, val32, or val64</b>	OUT	Data read from bus (8 bits for <code>viPeek8()</code> , 16 bits for <code>viPeek16()</code> , 32 bits for <code>viPeek32()</code> , and 64 bits for <code>viPeek64()</code> ).

## Return Values

None

## Description

The `viPeekXX()` operations read an 8-bit, 16-bit, 32-bit value, or 64-bit value, respectively, from the address location specified in `addr`. The address must be a valid memory address in the current process mapped by a previous `viMapAddress()` call.



**Note** If you use NI I/O Trace to debug these operations, enable the **Force peek/poke calls to appear in NI I/O Trace** option in Measurement & Automation Explorer (Windows), `visaconf` (Linux), or NI-VISA Configuration (Mac OS X). If you do not enable this option, NI I/O Trace might not log these operations.

## Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[VI\\_ATTR\\_WIN\\_ACCESS](#)

[viMapAddress/viMapAddressEx](#)

viPoke8/viPoke16/viPoke32/viPoke64

## viPoke8/viPoke16/viPoke32/viPoke64

## Purpose

Writes an 8-bit, 16-bit, 32-bit, or 64-bit value to the specified address.

## C Syntax

```
void viPoke8(ViSession vi, ViAddr addr, ViUInt8 val8)
```

```
void viPoke16(ViSession vi, ViAddr addr, ViUInt16 val16)
```

```
void viPoke32(ViSession vi, ViAddr addr, ViUInt32 val32)
```

```
void viPoke64(ViSession vi, ViAddr addr, ViUInt64 val64)
```

## Visual Basic Syntax

```
viPoke8(ByVal vi&, ByVal addr&, ByVal val8 as Byte)
```

```
viPoke16(ByVal vi&, ByVal addr&, ByVal val16%)
```

```
viPoke32(ByVal vi&, ByVal addr&, ByVal val32&)
```

## Resource Classes

PXI INSTR, PXI MEMACC, VXI INSTR, VXI MEMACC

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.




<b>addr</b>	IN	Destination address to store the value.
<b>val8, val16, val32, or val64</b>	IN	Value to be stored (8 bits for <code>viPoke8()</code> , 16 bits for <code>viPoke16()</code> , 32 bits for <code>viPoke32()</code> , 64 bits for <code>viPoke64()</code> ).

## Return Values

None

## Description

The `viPokeXX()` operations store the content of an 8-bit, 16-bit, 32-bit value, or 64-bit value, respectively, to the address pointed to by `addr`. The address must be a valid memory address in the current process mapped by a previous `viMapAddress()` call.



**Note** If you use NI I/O Trace to debug these operations, enable the **Force peek/poke calls to appear in NI I/O Trace** option in Measurement & Automation Explorer (Windows), `visaconf` (Linux), or NI-VISA Configuration (Mac OS X). If you do not enable this option, NI I/O Trace might not log these operations.

---

## Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[VI\\_ATTR\\_WIN\\_ACCESS](#)

[viMapAddress/viMapAddressEx](#)

[viPeek8/viPeek16/viPeek32/viPeek64](#)

# viPrintf

## Purpose

Converts, formats, and sends the parameters (designated by...) to the device as specified by the format string.

## C Syntax

```
ViStatus viPrintf(ViSession vi, ViString writeFmt, ...)
```

## Visual Basic Syntax

N/A

## Resource Classes

GPIB INSTR, GPIB INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

## Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
writeFmt	IN	String describing the format for arguments.
...	IN	Parameters to which the format string is applied.

## Return Values

Completion Codes	Description
------------------	-------------

VI_SUCCESS	Parameters were successfully formatted.
------------	---

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by <b>vi</b> has been locked for this kind of access.
VI_ERROR_IO	Could not perform write operation because of I/O error.
VI_ERROR_TMO	Timeout expired before write operation completed.
VI_ERROR_INV_FMT	A format specifier in the <b>writeFmt</b> string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the <b>writeFmt</b> string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

## Description

The `viPrintf()` operation sends data to a device as specified by the format string. Before sending the data, the operation formats the arguments in the parameter list as specified in the **writeFmt** string. The `viWrite()` operation performs the actual low-level I/O to the device. As a result, you should not use the `viWrite()` and `viPrint`

`f()` operations in the same session.

The **writeFmt** string can include regular character sequences, special formatting characters, and special format specifiers. The regular characters (including white spaces) are written to the device unchanged. The special characters consist of '\' (backslash) followed by a character. The format specifier sequence consists of '%' (percent) followed by an optional modifier (flag), followed by a format code.

## Special Formatting Characters

Special formatting character sequences send special characters. The following table lists the special characters and describes what they send to the device.

Formatting Character	Character Sent to Device
<code>\n</code>	Sends the ASCII LF character. The END identifier will also be automatically sent.
<code>\r</code>	Sends an ASCII CR character.
<code>\t</code>	Sends an ASCII TAB character.
<code>\###</code>	Sends the ASCII character specified by the octal value.
<code>\x##</code>	Sends the ASCII character specified by the hexadecimal value.
<code>\"</code>	Sends the ASCII double-quote (") character.
<code>\\</code>	Sends a backslash (\\) character.

## Format Specifiers

The format specifiers convert the next parameter in the sequence according to the modifier and format code, after which the formatted data is written to the specified device. The format specifier takes the following syntax:

`%[modifiers]format code`

where **format code** specifies which data type the argument is represented in. Modifiers are optional codes that describe the target data.

In the following tables, a 'd' format code refers to all conversion codes of type **integer** ('d', 'i', 'o', 'u', 'x', 'X'), unless specified as %d only. Similarly, an 'f' format code refers to all conversion codes of type **float** ('f', 'e', 'E', 'g', 'G'), unless specified as %f only.

Every conversion command starts with the % character and ends with a conversion character (format code). Between the % character and the format code, the following modifiers can appear in the sequence.

### ANSI C Standard Modifiers

Modifier	Supported with Format Code	Description
An integer specifying <b>field width</b> .	d, f, s format codes	<p>This specifies the minimum field width of the converted argument. If an argument is shorter than the <b>field width</b>, it will be padded on the left (or the right if the - flag is present).</p> <p>Special case:</p> <p>For the @H, @Q, and @B flags, the <b>field width</b> includes the #H, #Q, and #B strings, respectively.</p> <p>An asterisk (*) may be present in lieu of a <b>field width</b> modifier, in which case an extra <b>arg</b> is used. This <b>arg</b> must be an integer representing the <b>field width</b>.</p>

<p>An integer specifying <b>precision</b>.</p>	<p>d, f, s format codes</p>	<p>The <b>precision</b> string consists of a string of decimal digits. A decimal point ( . ) must prefix the <b>precision</b> string. The <b>precision</b> string specifies the following:</p> <ol style="list-style-type: none"> <li>1. The minimum number of digits to appear for the @l, @H, @Q, and @B flags and the i, o, u, x, and X format codes.</li> <li>2. The maximum number of digits after the decimal point in case of f format codes.</li> <li>3. The maximum numbers of characters for the string (s) specifier.</li> <li>4. Maximum significant digits for g format code.</li> </ol> <p>An asterisk (*) may be present in lieu of a <b>precision</b> modifier, in which case an extra <b>arg</b> is used. This <b>arg</b> must be an integer representing the <b>precision</b> of a numeric field.</p>
<p>An argument length modifier.</p> <p>h, l, ll, L, z, and Z are legal values. (z and Z are not ANSI C standard modifiers.)</p>	<p>h (d, b, B format codes)</p> <p>l (d, f, b, B format codes)</p> <p>ll (d, b, B format codes)</p> <p>L (f format code)</p> <p>z (b, B format codes)</p> <p>Z (b, B format codes)</p>	<p>The argument length modifiers specify one of the following:</p> <ol style="list-style-type: none"> <li>1. The h modifier promotes the argument to a short or unsigned short, depending on the format code type.</li> <li>2. The l modifier promotes the argument to a long or unsigned long.</li> <li>3. The ll modifier promotes the argument to a long long or unsigned long long.</li> <li>4. The L modifier promotes the argument to a long double parameter.</li> <li>5. The z modifier promotes the argument to an array of floats.</li> <li>6. The Z modifier promotes the argument to an array of doubles.</li> </ol>

## Enhanced Modifiers to ANSI C Standards

Modifier	Supported with Format Code	Description
A comma (,) followed by an integer <b><i>n</i></b> , where <b><i>n</i></b> represents the array size.	%d (plus variants) and %f only	<p>The corresponding argument is interpreted as a reference to the first element of an array of size <b><i>n</i></b>. The first <b><i>n</i></b> elements of this list are printed in the format specified by the format code.</p> <p>An asterisk (*) may be present after the <b><i>comma</i></b> (,) modifier, in which case an extra <b><i>arg</i></b> is used. This <b><i>arg</i></b> must be an integer representing the array size of the given type.</p>
@1	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined NR1 compatible number, which is an integer without any decimal point (for example, 123).
@2	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined NR2 compatible number. The NR2 number has at least one digit after the decimal point (for example, 123.45).
@3	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined NR3 compatible number. An NR3 number is a floating point number represented in an exponential form (for example, 1.2345E-67).
@H	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined <HEXADECIMAL NUMERIC RESPONSE DATA>. The number is represented in a base of sixteen form. Only capital letters should represent numbers. The number is of form #HXXX., where XXX. is a hexadecimal number (for example, #HAF35B).

@Q	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined <OCTAL NUMERIC RESPONSE DATA>. The number is represented in a base of eight form. The number is of the form #QYYY.., where YYY.. is an octal number (for example, #Q71234).
@B	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined <BINARY NUMERIC RESPONSE DATA>. The number is represented in a base two form. The number is of the form #BZZZ.., where ZZZ.. is a binary number (for example, #B011101001).

The following are the allowed format code characters. A format specifier sequence should include one and only one format code.

ANSI C Standard Format Codes

- % Send the ASCII percent (%) character.
- c Argument type: A character to be sent.
- d Argument type: An integer.

Modifier	Interpretation
Default functionality	Print an integer in NR1 format (an integer without a decimal point).
@2 or @3	The integer is converted into a floating point number and output in the correct format.
<b>field width</b>	Minimum field width of the output number. Any of the six IEEE 488.2 modifiers can also be specified with <b>field width</b> .



Length modifier l	<b>arg</b> is a long integer.
Length modifier ll	<b>arg</b> is a long long integer.
Length modifier h	<b>arg</b> is a short integer.
<b>, array size</b>	<b>arg</b> points to an array of integers (or long or short integers, depending on the length modifier) of size array size. The elements of this array are separated by array size - 1 commas and output in the specified format.

£ Argument type: A floating point number.

Modifier	Interpretation
Default functionality	Print a floating point number in NR2 format (a number with at least one digit after the decimal point).
@1	Print an integer in NR1 format. The number is truncated.
@3	Print a floating point number in NR3 format (scientific notation). <b>Precision</b> can also be specified.
<b>field width</b>	Minimum field width of the output number. Any of the six IEEE 488.2 modifiers can also be specified with <b>field width</b> .
Length modifier l	<b>arg</b> is a double float.

Length modifier $\mathbb{L}$	<b>arg</b> is a long double.
<b>, array size</b>	<b>arg</b> points to an array of floats (or doubles or long doubles, depending on the length modifier) of size array size. The elements of this array are separated by array size - 1 commas and output in the specified format.

s Argument type: A reference to a NULL-terminated string that is sent to the device without change.

## Enhanced Format Codes

b Argument type: A location of a block of data.

Flag or Modifier	Interpretation
Default functionality	The data block is sent as an IEEE 488.2 <DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA>. A count (long integer) must appear as a flag that specifies the number of elements (by default, bytes) in the block. A <b>field width</b> or <b>precision</b> modifier is not allowed with this format code.
* (asterisk)	An asterisk may be present instead of the count. In such a case, two <b>args</b> are used, the first of which is a long integer specifying the count of the number of elements in the data block. The second <b>arg</b> is a reference to the data block. The size of an element is determined by the optional length modifier (see below), and the default is byte width.
Length modifier $\mathbb{h}$	<b>arg</b> points to an array of unsigned short integers (16 bits). The count corresponds to the number of words rather than bytes. The data is swapped and padded into standard IEEE 488.2 format, if native computer representation is different.

Length modifier <code>l</code>	<b>arg</b> points to an array of unsigned long integers. The count specifies the number of long words (32 bits). Each longword data is swapped and padded into standard IEEE 488.2 format, if native computer representation is different.
Length modifier <code>ll</code>	<b>arg</b> points to an array of unsigned long long integers. The count specifies the number of long long words (64 bits). Each longword data is swapped and padded into standard IEEE 488.2 format, if native computer representation is different.
Length modifier <code>z</code>	<b>arg</b> points to an array of floats. The count specifies the number of floating point numbers (32 bits). The numbers are represented in IEEE 754 format, if native computer representation is different.
Length modifier <code>z</code>	<b>arg</b> points to an array of doubles. The count specifies the number of double floats (64 bits). The numbers will be represented in IEEE 754 format, if native computer representation is different.

**B** Argument type: A location of a block of data. The functionality is similar to `b`, except the data block is sent as an IEEE 488.2 <INDEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA>. This format involves sending an ASCII LF character with the END indicator set after the last byte of the block.

The END indicator is not appended when `LF(\n)` is part of a binary data block, as with `%b` or `%B`.

**y** Argument type: A location of a block of binary data.

Modifier	Interpretation
Default functionality	The data block is sent as raw binary data. A count (long integer) must appear as a flag that specifies the number of elements (by default, bytes) in the block. A field width or precision modifier is not allowed with this format code.

* (asterisk)	An asterisk may be present instead of the count. In such a case, two <b>args</b> are used, the first of which is a long integer specifying the count of the number of elements in the data block. The second <b>arg</b> is a reference to the data block. The size of an element is determined by the optional length modifier (see below), and the default is byte width.
Length modifier <code>h</code>	<b>arg</b> points to an array of unsigned short integers (16 bits). The count corresponds to the number of words rather than bytes. If the optional <code>!o1</code> byte order modifier is present, the data is sent in little endian format; otherwise, the data is sent in standard IEEE 488.2 format. The data will be byte swapped and padded as appropriate if native computer representation is different.
Length modifier <code>l</code>	<b>arg</b> points to an array of unsigned long integers (32 bits). The count specifies the number of long words rather than bytes. If the optional <code>!o1</code> byte order modifier is present, the data is sent in little endian format; otherwise, the data is sent in standard IEEE 488.2 format. The data will be byte swapped and padded as appropriate if native computer representation is different.
Length modifier <code>ll</code>	<b>arg</b> points to an array of unsigned long long integers (64 bits). The count specifies the number of long long words rather than bytes. If the optional <code>!o1</code> byte order modifier is present, the data is sent in little endian format; otherwise, the data is sent in standard IEEE 488.2 format. The data will be byte swapped and padded as appropriate if native computer representation is different.
Length modifier <code>z</code>	<b>arg</b> points to an array of floats. The count specifies the number of floating-point numbers (32 bits). If the optional <code>!o1</code> modifier is present, the data is sent in little endian format; otherwise, the data is sent in standard IEEE 488.2 format. The data will be byte swapped and padded as appropriate to native computer format.

Length modifier <code>z</code>	<code>arg</code> points to an array of doubles. The count specifies the number of double floats (64 bits). If the optional <code>!ol</code> modifier is present, the data is sent in little endian format; otherwise, the data is sent in standard IEEE 488.2 format. The data will be byte swapped and padded as appropriate to native computer format.
Byte order modifier <code>!ob</code>	Data is sent in standard IEEE 488.2 (big endian) format. This is the default behavior if neither <code>!ob</code> nor <code>!ol</code> is present.
Byte order modifier <code>!ol</code>	Data is sent in little endian format.

## Other ANSI C Conversion Codes

For ANSI C compatibility, VISA also supports the following conversion codes for output codes: `'i'`, `'o'`, `'u'`, `'n'`, `'x'`, `'X'`, `'e'`, `'E'`, `'g'`, `'G'`, and `'p'`. For further explanation of these conversion codes, see the ANSI C Standard.

Also refer to your ANSI C documentation for information on the `printf` function.



**Note** VISA will not send out the data across the bus, by default, until a `'\n'` character is encountered in the format string (not the data stream). You can modify this behavior with the `VI_ATTR_WR_BUF_OPER_MODE` attribute or with the `viFlush()` operation.

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[VI\\_ATTR\\_WR\\_BUF\\_OPER\\_MODE](#)

[viFlush](#)

[viScanf](#)

[viSprintf](#)

[viVPrintf](#)

[viVSPrintf](#)

# viQueryf

## Purpose

Performs a formatted write and read through a single call to an operation.

## C Syntax

```
ViStatus viQueryf(ViSession vi, ViString writeFmt, ViString readFmt, ...)
```

## Visual Basic Syntax

N/A

## Resource Classes

GPIO INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR

## Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.

<b>writeFmt</b>	IN	String describing the format of write arguments.
<b>readFmt</b>	IN	String describing the format of read arguments.
...	IN/OUT	Parameters to which write and read format strings are applied.

## Return Values

Completion Codes	Description
VI_SUCCESS	Successfully completed the query operation.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_IO	Could not perform Read/Write operation because of I/O error.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by <b>vi</b> has been locked for this kind of access.
VI_ERROR_TMO	Timeout occurred before Read/Write operation completed.
VI_ERROR_INV_FMT	A format specifier in the <b>writeFmt</b> or <b>readFmt</b> string is

	invalid.
VI_ERROR_NSUP_FMT	The format specifier is not supported for current argument type.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

## Description

This operation provides a mechanism of ***Send, then receive*** typical to a command sequence from a commander device. In this manner, the response generated from the command can be read immediately.

This operation is a combination of the `viPrintf()` and `viScanf()` operations. The first ***n*** arguments corresponding to the first format string are formatted by using the **writeFmt** string, then sent to the device. The write buffer is flushed immediately after the write portion of the operation completes. After these actions, the response data is read from the device into the remaining parameters (starting from parameter ***n*** + 1) using the **readFmt** string.



**Note** Because the prototype for this function cannot provide complete type-checking, remember that all output parameters must be passed by reference.

---

## Related Topics

[INSTR Resource](#)

[SOCKET Resource](#)

[viPrintf](#)



[viScanf](#)

[viVQueryf](#)

# viRead

## Purpose

Reads data from device or interface synchronously.

## C Syntax

```
ViStatus viRead(ViSession vi, ViPBuf buf, ViUInt32 count, ViPU
Int32 retCount)
```

## Visual Basic Syntax

```
viRead&(ByVal vi&, ByVal buf$, ByVal count&, retCount&)
```

## Resource Classes

GPIO INSTR, GPIO INTRFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB
RAW, VXI INSTR, VXI SERVANT

## Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
buf	OUT	Location of a buffer to receive data from device.
count	IN	Number of bytes to be read.

<b>retCount</b>	OUT	Number of bytes actually transferred.
-----------------	-----	---------------------------------------

## Return Values

Completion Codes	Description
VI_SUCCESS	The operation completed successfully and the END indicator was received (for interfaces that have END indicators). This completion code is returned regardless of whether the termination character is received or the number of bytes read is equal to <b>count</b> .
VI_SUCCESS_TERM_CHAR	The specified termination character was read but no END indicator was received. This completion code is returned regardless of whether the number of bytes read is equal to <b>count</b> .
VI_SUCCESS_MAX_CNT	The number of bytes read is equal to <b>count</b> . No END indicator was received and no termination character was read.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given <b>vi</b> does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by <b>vi</b> has been locked for this kind of

	access.
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_INV_SETUP	Unable to start read operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_NCIC	The interface associated with the given <code>vi</code> is not currently the controller in charge.
VI_ERROR_NLISTENERS	No-listeners condition is detected (both <code>NRFD</code> and <code>NDAC</code> are unasserted).
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.
<code>VI_ERROR_IO</code>	An unknown I/O error occurred during transfer.
<code>VI_ERROR_CONN_LOST</code>	The I/O connection for the given session has been lost.

## Description

The `viRead()` 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.



**Note** The `retCount` and `buf` parameters always are valid on both success and error.

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[viBufRead](#)

[viReadAsync](#)

[viReadToFile](#)

[viWrite](#)

# viReadAsync

## Purpose

Reads data from device or interface asynchronously.

## C Syntax

```
ViStatus viReadAsync(ViSession vi, ViPBuf buf, ViUInt32 count,  
ViPJobId jobId)
```

## Visual Basic Syntax

N/A

## Resource Classes

GPIO INSTR, GPIB INTRFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>buf</b>	OUT	Location of a buffer to receive data from device.
<b>count</b>	IN	Number of bytes to be read.
<b>jobId</b>	OUT	Job ID of this asynchronous read operation.

## Return Values

Completion Codes	Description
VI_SUCCESS	Asynchronous read operation successfully queued.
VI_SUCCESS_SYNC	Read operation performed synchronously.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by <code>vi</code> has been locked for this kind of access.
VI_ERROR_QUEUE_ERROR	Unable to queue read operation (usually due to the I/O completion event not being enabled or insufficient space in the session's queue).
VI_ERROR_IN_PROGRESS	Unable to queue the asynchronous operation because there is already an operation in progress.

## Description

The `viReadAsync()` operation asynchronously transfers data. The data read is to be stored in the buffer represented by **buf**. This operation normally returns before the transfer terminates.

Before calling this operation, you should enable the session for receiving I/O

completion events. After the transfer has completed, an I/O completion event is posted.

The operation returns **jobId**, which you can use with either `viTerminate()` to abort the operation, or with an I/O completion event to identify which asynchronous read operation completed. VISA will never return `VI_NULL` for a valid **jobId**.



**Note** If you have enabled `VI_EVENT_IO_COMPLETION` for queueing (`VI_QUEUE`), for each successful call to `viReadAsync()`, you must call `viWaitOnEvent()` to retrieve the I/O completion event. This is true even if the I/O is done synchronously (that is, if the operation returns `VI_SUCCESS_SYNC`). If you are using LabVIEW, this is done for you automatically.

---

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[VI\\_EVENT\\_IO\\_COMPLETION](#)

[viEnableEvent](#)

[viRead](#)

[viTerminate](#)

[viWaitOnEvent](#)

[viWriteAsync](#)

**viReadSTB**

## Purpose

Reads a status byte of the service request.

## C Syntax

```
ViStatus viReadSTB(ViSession vi, ViPUInt16 status)
```

## Visual Basic Syntax

```
viReadSTB&(ByVal vi&, status%)
```

## Resource Classes

GPIB INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR

## Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.
status	OUT	Service request status byte.

## Return Values

Completion Codes	Description
VI_SUCCESS	The operation completed successfully.

Error Codes	Description
-------------	-------------



VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_SRQ_NOCCURRED	Service request has not been received for the session.
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_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 unasserted).
VI_ERROR_INV_SETUP	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).

<code>VI_ERROR_CONN_LOST</code>	The I/O connection for the given session has been lost.
---------------------------------	---

## Description

### Status Bytes for 488.2 Instruments (GPIB, VXI, TCPIP, and USB)

This operation reads a service request status from a message-based device. The bus-specific details are:

- For a GPIB device, the status is read by serial polling the device.
- For a VXI device, VISA sends the Word Serial Read STB query.
- For a USB device, this function sends the `READ_STATUS_BYTE` command on the control pipe.

### Status Bytes for Non-488.2 Instruments (Serial INSTR, TCPIP SOCKET, and USB RAW)

A message is sent in response to a service request to retrieve status information. If `VI_ATTR_IO_PROT` is `VI_PROT_4882_STRS`, the device is sent the string `"*STB?\n"`, and then the device's status byte is read; otherwise, this operation is not valid.

Although the **status** output is a 16-bit value, the upper 8 bits are always 0. The lower 8 bits contain the actual status byte. For 488.2 instruments, this is the 488.2-defined status byte.

The IEEE 488.2 standard defines several bit assignments in the status byte. For example, if bit 6 of the **status** is set, the device is requesting service. In addition to setting bit 6 when requesting service, 488.2 devices also use two other bits to specify their status. Bit 4, the Message Available bit (MAV), is set when the device is ready to send previously queried data. Bit 5, the Event Status bit (ESB), is set if one or more of the enabled 488.2 events occurs. These events include power-on, user request, command error, execution error, device dependent error, query error, request control, and operation complete. The device can assert SRQ when ESB or MAV are set, or when a manufacturer-defined condition occurs. Manufacturers of 488.2 devices use the remaining lower-order bits to communicate the reason for the service request or to summarize the device state.

## Related Topics

[INSTR Resource](#)

[SOCKET Resource](#)

[VI\\_ATTR\\_IO\\_PROT](#)

[VI\\_EVENT\\_SERVICE\\_REQ](#)

## viReadToFile

### Purpose

Read data synchronously, and store the transferred data in a file.

### C Syntax

```
ViStatus viReadToFile(ViSession vi, ViString fileName, ViUInt32 count, ViPUInt32 retCount)
```

### Visual Basic Syntax

```
viReadToFile& (ByVal vi&, ByVal filename$, ByVal count&, retCount&)
```

### Resource Classes

GPIO INSTR, GPIO INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

### Parameters

Name	Direction	Description
------	-----------	-------------

<b>vi</b>	IN	Unique logical identifier to a session.
<b>fileName</b>	IN	Name of file to which data will be written.
<b>count</b>	IN	Number of bytes to be read.
<b>retCount</b>	OUT	Number of bytes actually transferred.

## Return Values

Completion Codes	Description
VI_SUCCESS	The operation completed successfully and the END indicator was received (for interfaces that have END indicators).
VI_SUCCESS_TERM_CHAR	The specified termination character was read.
VI_SUCCESS_MAX_CNT	The number of bytes read is equal to count.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session or object reference is invalid (both are the same value).
VI_ERROR_NSUP_OPER	The given vi does not support this operation.

VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by <code>vi</code> has been locked for this kind of access.
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_INV_SETUP	Unable to start read operation because setup is invalid (due to attributes being set to an inconsistent state).
VI_ERROR_NCIC	The interface associated with the given <code>vi</code> is not currently the controller in charge.
VI_ERROR_NLISTENERS	No listeners condition is detected (both <code>NRFD</code> and <code>NDAC</code> are deasserted).
VI_ERROR_ASRL_PARITY	A parity error occurred during transfer.
VI_ERROR_ASRL_FRAMING	A framing error occurred during transfer.

<code>VI_ERROR_ASRL_OVERRUN</code>	An overrun error occurred during transfer. A character was not read from the hardware before the next character arrived.
<code>VI_ERROR_IO</code>	An unknown I/O error occurred during transfer.
<code>VI_ERROR_FILE_ACCESS</code>	An error occurred while trying to open the specified file. Possible reasons include an invalid path or lack of access rights.
<code>VI_ERROR_FILE_IO</code>	An error occurred while accessing the specified file.
<code>VI_ERROR_CONN_LOST</code>	The I/O connection for the given session has been lost.

## Description

This read operation synchronously transfers data. The file specified in `fileName` is opened in binary write-only mode. If the value of `VI_ATTR_FILE_APPEND_EN` is `VI_FALSE`, any existing contents are destroyed; otherwise, the file contents are preserved. The data read is written to the file. This operation returns only when the transfer terminates.

This operation is useful for storing raw data to be processed later.

### Special Values for `retCount` Parameter

Value	Action Description
<code>VI_NULL</code>	Do not return the number of bytes transferred.



**Note** The `retCount` parameter always is valid on both success and error.

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[VI\\_ATTR\\_FILE\\_APPEND\\_EN](#)

[viRead](#)

[viWriteFromFile](#)

## viScanf

### Purpose

Reads, converts, and formats data using the format specifier. Stores the formatted data in the parameters (designated by ...).

### C Syntax

```
ViStatus viScanf(ViSession vi, ViString readFmt, ...)
```

### Visual Basic Syntax

N/A

### Resource Classes

GPIO INSTR, GPIO INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>readFmt</b>	IN	String describing the format for arguments.
...	OUT	Parameters into which the data is read and the format string is applied.

## Return Values

Completion Codes	Description
VI_SUCCESS	Data was successfully read and formatted into ... parameter(s).

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
VI_ERROR_IO	Could not perform read operation because of I/O error.



<code>VI_ERROR_TMO</code>	Timeout expired before read operation completed.
<code>VI_ERROR_INV_FMT</code>	A format specifier in the <b>readFmt</b> string is invalid.
<code>VI_ERROR_NSUP_FMT</code>	A format specifier in the <b>readFmt</b> string is not supported.
<code>VI_ERROR_ALLOC</code>	The system could not allocate a formatted I/O buffer because of insufficient resources.

## Description

The `viScanf()` operation receives data from a device, formats it by using the format string, and stores the resulting data in the **arg** parameter list. The `viRead()` operation is used for the actual low-level read from the device. As a result, you should not use the `viRead()` and `viScanf()` operations in the same session.



**Note** Because the prototype for this function cannot provide complete type-checking, remember that all output parameters must be passed by reference.

The format string can have format specifier sequences, white characters, and ordinary characters. The white characters—blank, vertical tabs, horizontal tabs, form feeds, new line/linefeed, and carriage return—are ignored except in the case of `%c` and  `%[ ]`. All other ordinary characters except `%` should match the next character read from the device.

The format string consists of a `%`, followed by optional modifier flags, followed by one of the format codes in that sequence. It is of the form:

`%[modifier]format code`

where the optional modifier describes the data format, while format code indicates the nature of data (data type). One and only one format code should be performed at the specifier sequence. A format specification directs the conversion to the next input

**arg.**

The results of the conversion are placed in the variable that the corresponding argument points to, unless the \* assignment-suppressing character is given. In such a case, no **arg** is used and the results are ignored.


The `viScanf()` operation accepts input until an END indicator is read or all the format specifiers in the **readFmt** string are satisfied. Thus, detecting an END indicator before the **readFmt** string is fully consumed will result in ignoring the rest of the format string. Also, if some data remains in the buffer after all format specifiers in the **readFmt** string are satisfied, the data will be kept in the buffer and will be used by the next `viScanf()` operation.

When `viScanf()` times out, the next call to `viScanf()` will read from an empty buffer and force a read from the device.

Notice that when an END indicator is received, not all arguments in the format string may be consumed. However, the operation still returns a successful completion code.

The following two tables describe optional modifiers that can be used in a format specifier sequence.

ANSI C Standard Modifiers

Modifier	Supported with Format Code	Description
An integer representing the <b>field width</b>	%s, %c, %[] format codes	<div>It specifies the maximum field width that the argument will take. A '#' may also appear instead of the integer <b>field width</b>, in which case the next <b>arg</b> is a reference to the <b>field width</b>. This <b>arg</b> is a reference to an integer for %c and %s. The <b>field width</b> is not allowed for %d or %f.</div> <div> <b>Note</b> The # modifier is a VISA extension.</div>
A length		The argument length modifiers specify one of the

<p>modifier ('h', 'l', 'll', 'L', 'z', or 'Z'). z and Z are not ANSI C standard modifiers.</p>	<p>h (d, b format codes)</p> <p>l (d, f, b format codes)</p> <p>ll (d and b format codes)</p> <p>L (f format code)</p> <p>z (b format code)</p> <p>Z (b format code)</p>	<p>following:</p> <ol style="list-style-type: none"> <li>1. The h modifier promotes the argument to be a reference to a short integer or unsigned short integer, depending on the format code.</li> <li>2. The l modifier promotes the argument to point to a long integer or unsigned long integer.</li> <li>3. The ll modifier promotes the argument to a long long or unsigned long long.</li> <li>4. The L modifier promotes the argument to point to a long double floats parameter.</li> <li>5. The z modifier promotes the argument to point to an array of floats.</li> <li>6. The Z modifier promotes the argument to point to an array of double floats.</li> </ol>
*	All format codes	<p>An asterisk (*) acts as the assignment suppression character. The input is not assigned to any parameters and is discarded.</p>

## Enhanced Modifiers to ANSI C Standards

Modifier	Supported with Format Code	Description
<p>A comma (,) followed by an integer <b>n</b>, where <b>n</b> represents the array size.</p>	%d (plus variants) and %f only	<p>The corresponding argument is interpreted as a reference to the first element of an array of size <b>n</b>. The first <b>n</b> elements of this list are printed in the format specified by the format code.</p> <p>A number sign (#) may be present after the comma (,) modifier, in which case an extra <b>arg</b> is used. This <b>arg</b> must be an integer representing the array size of the given type.</p>
@1	%d (plus variants) and %f only	<p>Converts to an IEEE 488.2 defined NR1 compatible number, which is an integer without any decimal point</p>

		(for example, 123).
@2	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined NR2 compatible number. The NR2 number has at least one digit after the decimal point (for example, 123.45).
@H	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined <HEXADECIMAL NUMERIC RESPONSE DATA>. The number is represented in a base of sixteen form. Only capital letters should represent numbers. The number is of form #HXXX., where XXX. is a hexadecimal number (for example, #HAF35B).
@Q	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined <OCTAL NUMERIC RESPONSE DATA>. The number is represented in a base of eight form. The number is of the form #QYYY., where YY Y. is an octal number (for example, #Q71234).
@B	%d (plus variants) and %f only	Converts to an IEEE 488.2 defined <BINARY NUMERIC RESPONSE DATA>. The number is represented in a base two form. The number is of the form #BZZZ., where ZZ Z. is a binary number (for example, #B011101001).

## ANSI C Standard Format Codes

c Argument type: A reference to a character.

Flags or Modifiers	Interpretation
Default functionality	A character is read from the device and stored in the parameter.

<b><i>field width</i></b>	<b><i>field width</i></b> number of characters are read and stored at the reference location (the default <b><i>field width</i></b> is 1). No NULL character is added at the end of the data block.
---------------------------	---



**Note** This format code does not ignore white space in the device input stream.

d Argument type: A reference to an integer.

Flags or Modifiers	Interpretation
Default functionality	Characters are read from the device until an entire number is read. The number read may be in either IEEE 488.2 formats <DECIMAL NUMERIC PROGRAM DATA>, also known as NRf; flexible numeric representation (NR1, NR2, NR3...); or <NON-DECIMAL NUMERIC PROGRAM DATA> (#H, #Q, and #B).
<b><i>field width</i></b>	The input number will be stored in a field at least this wide.
Length modifier l	<b>arg</b> is a reference to a long integer.
Length modifier ll	<b>arg</b> is a reference to a long long integer.
Length modifier h	<b>arg</b> is a reference to a short integer. Rounding is performed according to IEEE 488.2 rules (0.5 and up).
<b><i>, array size</i></b>	<b>arg</b> points to an array of integers (or long or short integers, depending on the length modifier) of size array size. The elements of this array should be separated by commas. Elements will be read until either array size number of elements are consumed or they are no longer separated by commas. If the <b><i>array size</i></b> contains a number sign (#),

	two arguments are used. The first <b>arg</b> read is a pointer to an integer specifying the maximum number of elements that the array can hold. The second <b>arg</b> should be a reference to an array. Also, the actual number of elements read is stored back in the first argument.
--	---

ƒ Argument type: A reference to a floating point number.

Flags or Modifiers	Interpretation
Default functionality	Characters are read from the device until an entire number is read. The number read may be in either IEEE 488.2 formats <DECIMAL NUMERIC PROGRAM DATA> (NRf) or <NON-DECIMAL NUMERIC PROGRAM DATA> (#H, #Q, and #B)
<b>field width</b>	The input will be stored in a field at least this wide.
Length modifier $\mathbb{L}$	<b>arg</b> is a reference to a double floating point number.
Length modifier $\mathbb{L}$	<b>arg</b> is a reference to a long double number.
<b>, array size</b>	<b>arg</b> points to an array of floats (or double or long double, depending on the length modifier) of size array size. The elements of this array should be separated by commas. Elements will be read until either array size number of elements are consumed or they are no longer separated by commas. If the <b>array size</b> contains a number sign (#), two arguments are used. The first <b>arg</b> read is a pointer to an integer specifying the maximum number of elements that the array can hold. The second <b>arg</b> should be a reference to an array. Also, the actual number of elements read is stored back in the first argument.

s Argument type: A reference to a string.

Flags or Modifiers	Interpretation
Default functionality	All leading white space characters are ignored. Characters are read from the device into the string until a white space character is read.
<b><i>field width</i></b>	This flag gives the maximum string size. If the <b><i>field width</i></b> contains a number sign (#), two arguments are used. The first argument read is a pointer to an integer specifying the maximum array size. The second should be a reference to an array. In case of <b><i>field width</i></b> characters already read before encountering a white space, additional characters are read and discarded until a white space character is found. In case of <b><i># field width</i></b> , the actual number of characters that were copied into the user array, not counting the trailing NULL character, are stored back in the integer pointed to by the first argument.

## Enhanced Format Codes

b Argument type: A reference to a data array.

Flags or Modifiers	Interpretation
Default functionality	The data must be in IEEE 488.2 <ARBITRARY BLOCK PROGRAM DATA> format. The format specifier sequence should have a flag describing the <b><i>field width</i></b> , which will give a maximum count of the number of bytes (or words or longwords, depending on length modifiers) to be read from the device. If the <b><i>field width</i></b> contains a # sign, two arguments are used. The first <b>arg</b> read is a pointer to a long integer specifying the maximum number of elements that the array can hold. The second <b>arg</b> should be a reference to an array. Also, the actual number of elements read is stored back in the first argument. In absence of length modifiers, the data is assumed to be of byte-size elements. In some cases, data might be read until an END indicator is read.
Length modifier h	<b>arg</b> points to an array of 16-bit words, and count specifies the number

	of words. Data that is read is assumed to be in IEEE 488.2 byte ordering. It will be byte swapped and padded as appropriate to native computer format.
Length modifier $\mathbb{1}$	<b>arg</b> points to an array of 32-bit long words, and count specifies the number of long words. Data that is read is assumed to be in IEEE 488.2 byte ordering. It will be byte swapped and padded as appropriate to native computer format.
Length modifier $\mathbb{11}$	<b>arg</b> points to an array of 64-bit long long words, and count specifies the number of long long words. Data that is read is assumed to be in IEEE 488.2 byte ordering. It will be byte swapped and padded as appropriate to native computer format.
Length modifier $\mathbb{z}$	<b>arg</b> points to an array of floats, and count specifies the number of floating point numbers. Data that is read is an array of 32-bit IEEE 754 format floating point numbers.
Length modifier $\mathbb{Z}$	<b>arg</b> points to an array of doubles, and the count specifies the number of floating point numbers. Data that is read is an array of 64-bit IEEE 754 format floating point numbers.

$\mathbb{t}$  Argument type: A reference to a string.

Flags or Modifiers	Interpretation
Default functionality	Characters are read from the device until the first END indicator is received. The character on which the END indicator was received is included in the buffer.
<b><i>field width</i></b>	This flag gives the maximum string size. If an END indicator is not



	received before <b>field width</b> number of characters, additional characters are read and discarded until an END indicator arrives. # <b>field width</b> has the same meaning as in %s.
--	---

T Argument type: A reference to a string.

Flags or Modifiers	Interpretation
Default functionality	Characters are read from the device until the first linefeed character ( <code>\n</code> ) is received. The linefeed character is included in the buffer.
<b>field width</b>	This flag gives the maximum string size. If a linefeed character is not received before <b>field width</b> number of characters, additional characters are read and discarded until a linefeed character arrives. # <b>field width</b> has the same meaning as in %s.

y Argument type: A location of a block of binary data.

Modifier	Interpretation
Default functionality	The data block is read as raw binary data. The format specifier sequence should have a flag describing the array size, which will give a maximum count of the number of bytes (or words or longwords, depending on length modifiers) to be read from the device. If the array size contains a # sign, two arguments are used. The first argument read is a pointer to a long integer that specifies the maximum number of elements that the array can hold. The second argument should be a reference to an array. Also, the actual number of elements read is stored back in the first argument. In absence of length modifiers, the data is assumed to be byte-size elements. In some cases, data might be read until an END indicator is read.

Length modifier <code>h</code>	The data block is assumed to be a reference to an array of unsigned short integers (16 bits). The count corresponds to the number of words rather than bytes. If the optional <code>!o1</code> modifier is present, the data read is assumed to be in little endian format; otherwise, the data read is assumed to be in standard IEEE 488.2 format. The data will be byte swapped and padded as appropriate to native computer format.
Length modifier <code>l</code>	The data block is assumed to be a reference to an array of unsigned long integers (32 bits). The count corresponds to the number of longwords rather than bytes. If the optional <code>!o1</code> modifier is present, the data read is assumed to be in little endian format; otherwise, the data read is assumed to be in standard IEEE 488.2 format. The data will be byte swapped and padded as appropriate to native computer format.
Length modifier <code>z</code>	The data block is assumed to be a reference to an array of single-precision floating-point numbers (32 bits). The count corresponds to the number of floats rather than bytes. If the optional <code>!o1</code> modifier is present, the data read is assumed to be in little endian format; otherwise, the data read is assumed to be in standard IEEE 488.2 format. The data will be byte swapped and padded as appropriate to native computer format.
Length modifier <code>Z</code>	The data block is assumed to be a reference to an array of double-precision floating-point numbers (64 bits). The count corresponds to the number of double floats rather than bytes. If the optional <code>!o1</code> modifier is present, the data read is assumed to be in little endian format; otherwise, the data read is assumed to be in standard IEEE 488.2 format. The data will be byte swapped and padded as appropriate to native computer format.
Byte order modifier <code>!ob</code>	The data being read is assumed to be in standard IEEE 488.2 (big endian) format. This is the default behavior if neither <code>!ob</code> nor <code>!o1</code> is present.

Byte order modifier !o1	The data being read is assumed to be in little endian format.
-------------------------	---

## Other ANSI C Format Specifiers

For ANSI C compatibility, VISA also supports the following format specifiers for input codes: 'i', 'o', 'u', 'n', 'x', 'X', 'e', 'E', 'g', 'G', 'p', '[...]', and '[^...]'. For further explanation of these conversion codes, see the ANSI C Standard.

---

### Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[VI\\_ATTR\\_RD\\_BUF\\_OPER\\_MODE](#)

[viFlush](#)

[viPrintf](#)

[viSScanf](#)

[viVScanf](#)

[viVSScanf](#)

[viSetAttribute](#)

# Purpose

Sets the state of an attribute.

# C Syntax

```
ViStatus viSetAttribute(ViObject vi, ViAttr attribute, ViAttrState attrState)
```

# Visual Basic Syntax

```
viSetAttribute&(ByVal vi&, ByVal attribute&, ByVal attrState&)
```

# Resource Classes

All I/O session types

# Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>attribute</b>	IN	Attribute for which the state is to be modified.
<b>attrState</b>	IN	The state of the attribute to be set for the specified object. The interpretation of the individual attribute value is defined by the object.

# Return Values

Completion Codes	Description
------------------	-------------

<code>VI_SUCCESS</code>	Attribute value set successfully.
<code>VI_WARN_NSUP_ATTR_STATE</code>	Although the specified attribute state is valid, it is not supported by this implementation.

Error Codes	Description
<code>VI_ERROR_INV_OBJECT</code>	The given object reference is invalid.
<code>VI_ERROR_NSUP_ATTR</code>	The specified attribute is not defined by the referenced object.
<code>VI_ERROR_NSUP_ATTR_STATE</code>	The specified state of the attribute is not valid, or is not supported as defined by the object.
<code>VI_ERROR_ATTR_READONLY</code>	The specified attribute is Read Only.
<code>VI_ERROR_RSRC_LOCKED</code>	Specified operation could not be performed because the resource identified by <code>vi</code> has been locked for this kind of access.

## Description

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

Both `VI_WARN_NSUP_ATTR_STATE` and `VI_ERROR_NSUP_ATTR_STATE` indicate that the specified attribute state is not supported. A resource normally returns the error code `VI_ERROR_NSUP_ATTR_STATE` when it cannot set a specified attribute

state. The completion code `VI_WARN_NSUP_ATTR_STATE` is intended to alert the application that although the specified optional attribute state is not supported, the application should not fail. One example is attempting to set an attribute value that would increase performance speeds. This is different than attempting to set an attribute value that specifies required but nonexistent hardware (such as specifying a VXI ECL trigger line when no hardware support exists) or a value that would change assumptions a resource might make about the way data is stored or formatted (such as byte order).

Some attributes documented as being generally Read/Write may at times be Read Only. This is usually the case when an attribute configures how the VISA driver receives events of a given type, and the event type associated with that attribute is currently enabled. Under these circumstances, calling `viSetAttribute` on that attribute returns `VI_ERROR_ATTR_READONLY`.

The error code `VI_ERROR_RSRC_LOCKED` is returned only if the specified attribute is Read/Write and Global, and the resource is locked by another session.

---

## Related Topics

[Attributes](#)

[viGetAttribute](#)

[VISA Resource Template](#)

## viSetBuf

### Purpose

Sets the size for the formatted I/O and/or low-level I/O communication buffer(s).

### C Syntax

```
ViStatus viSetBuf(ViSession vi, ViUInt16 mask, ViUInt32 size)
```

## Visual Basic Syntax

```
viSetBuf&(ByVal vi&, ByVal mask%, ByVal size&)
```

## Resource Classes

GPIB INSTR, GPIB INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>mask</b>	IN	Specifies the type of buffer.
<b>size</b>	IN	The size to be set for the specified buffer(s).

## Return Values

Completion Codes	Description
VI_SUCCESS	Buffer size set successfully.
VI_WARN_NSUP_BUF	The specified buffer is not supported.

Error Codes	Description
-------------	-------------

VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by <b>vi</b> has been locked for this kind of access.
VI_ERROR_ALLOC	The system could not allocate the buffer(s) of the specified <b>size</b> because of insufficient resources.
VI_ERROR_INV_MASK	The system cannot set the buffer for the given <b>mask</b> .

## Description

The `viSetBuf()` operation changes the buffer size of the read and/or write buffer for formatted I/O and/or serial communication. The **mask** parameter specifies the buffer for which to set the size. The **mask** parameter can specify multiple buffers by bit-ORing any of the following values together.

Flags	Interpretation
VI_READ_BUF (1)	Formatted I/O read buffer.
VI_WRITE_BUF (2)	Formatted I/O write buffer.
VI_IO_IN_BUF (16)	Low-level I/O receive buffer.
VI_IO_OUT_BUF (32)	Low-level I/O transmit buffer.

A call to `viSetBuf()` flushes the session's related Read/Write buffer(s). Although you



can explicitly flush the buffers by making a call to `viFlush()`, the buffers are flushed implicitly under some conditions. These conditions vary for the `viPrintf()` and `viScanf()` operations.

Since not all serial drivers support user-defined buffer sizes, it is possible that a specific implementation of VISA may not be able to control this feature. If an application requires a specific buffer size for performance reasons, but a specific implementation of VISA cannot guarantee that size, then it is recommended to use some form of handshaking to prevent overflow conditions.

In previous versions of VISA, `VI_IO_IN_BUF` was known as `VI_ASRL_IN_BUF` and `VI_IO_OUT_BUF` was known as `VI_ASRL_OUT_BUF`.

---

## Related Topics

[Automatically Flushing the Formatted I/O Buffers](#)

[Controlling the Serial I/O Buffers](#)

[Formatted I/O Read and Low-Level I/O Receive Buffers](#)

[Formatted I/O Write and Low-Level I/O Transmit Buffers](#)

[INSTR Resource](#)

[INTFC Resource](#)

[Manually Flushing the Formatted I/O Buffers](#)

[Recommendations for Using the VISA Buffers](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[VI\\_ATTR\\_RD\\_BUF\\_SIZE](#)

[VI\\_ATTR\\_WR\\_BUF\\_SIZE](#)

[viFlush](#)

[viPrintf](#)

[viScanf](#)

# viSprintf

## Purpose

Converts, formats, and sends the parameters (designated by...) to a user-specified buffer as specified by the format string.

## C Syntax

```
ViStatus viSprintf(ViSession vi, ViPBuf buf, ViString writeFm  
t, ...)
```

## Visual Basic Syntax

N/A

## Resource Classes

GPIB INSTR, GPIB INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>buf</b>	OUT	Buffer where data is to be written.

<b>writeFmt</b>	IN	The format string to apply to parameters in <code>ViVAList</code> .
...	IN	Parameters to which the format string is applied. The formatted data is written to the specified buf.

## Return Values

Completion Codes	Description
<code>VI_SUCCESS</code>	Parameters were successfully formatted.

Error Codes	Description
<code>VI_ERROR_INV_OBJECT</code>	The given session reference is invalid.
<code>VI_ERROR_RSRC_LOCKED</code>	Specified operation could not be performed because the resource identified by <code>vi</code> has been locked for this kind of access.
<code>VI_ERROR_INV_FMT</code>	A format specifier in the <b>writeFmt</b> string is invalid.
<code>VI_ERROR_NSUP_FMT</code>	A format specifier in the <b>writeFmt</b> string is not supported.
<code>VI_ERROR_ALLOC</code>	The system could not allocate a formatted I/O buffer because of insufficient resources.

## Description

The `viSPrintf()` operation is similar to `viPrintf()`, except that the output is not written to the device; it is written to the user-specified buffer. This output buffer will be NULL terminated.

If this operation outputs an END indicator before all the arguments are satisfied, then the rest of the **writeFmt** string is ignored and the buffer string is still terminated by a NULL.



**Note** The size of the **buf** parameter should be large enough to hold the formatted I/O contents plus the NULL termination character.

---

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[viPrintf](#)

[viSScanf](#)

[viVPrintf](#)

[viVSPrintf](#)

## viSScanf

## Purpose

Reads, converts, and formats data from a user-specified buffer using the format

specifier. Stores the formatted data in the parameters (designated by ...).

## C Syntax

```
ViStatus viSScanf(ViSession vi, ViBuf buf, ViString readFmt,
...)
```

## Visual Basic Syntax

N/A

## Resource Classes

GPIO INSTR, GPIO INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>buf</b>	IN	Buffer from which data is read and formatted.
<b>readFmt</b>	IN	String describing the format for arguments.
<b>...</b>	OUT	Parameters into which the data is read and the format string is applied.

## Return Values

Completion Codes	Description
------------------	-------------

VI_SUCCESS	Data was successfully read and formatted into ... parameter(s).
------------	---

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by <b>vi</b> has been locked for this kind of access.
VI_ERROR_INV_FMT	A format specifier in the <b>readFmt</b> string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the <b>readFmt</b> string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

## Description

The `viSScanf()` operation is similar to `viSscanf()`, except that the data is read from a user-specified buffer rather than from a device.

---

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[viScanf](#)

[viSprintf](#)

[viVScanf](#)

[viVSScanf](#)

# viStatusDesc

## Purpose

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

## C Syntax

```
ViStatus viStatusDesc(ViObject vi, ViStatus status, ViChar desc[])
```

## Visual Basic Syntax

```
viStatusDesc&(ByVal vi&, ByVal status&, ByVal desc$)
```

## Resource Classes

All I/O session types, all event object types, VISA Resource Manager

## Parameters

Name	Direction	Description
------	-----------	-------------


vi	IN	Unique logical identifier to a session.
status	IN	Status code to interpret.
desc	OUT	The user-readable string interpretation of the status code passed to the operation.

Return Values

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

Description

The `viStatusDesc()` operation is used to retrieve a user-readable string that describes the status code presented. If the string cannot be interpreted, the operation returns the warning code `VI_WARN_UNKNOWN_STATUS`. However, the output string `desc` is valid regardless of the status return value.



**Note** The size of the `desc` parameter should be ***at least*** 256 bytes.

Related Topics

[Completion Codes](#)

[Error Codes](#)



## VISA Resource Template

# viTerminate

## Purpose

Requests a VISA session to terminate normal execution of an operation.

## C Syntax

```
ViStatus viTerminate(ViObject vi, ViUInt16 degree, ViJobId jobId)
```

## Visual Basic Syntax

N/A

## Resource Classes

GPIO INSTR, GPIO INTFC, PXI INSTR, PXI MEMACC, PXI BACKPLANE, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI MEMACC, VXI BACKPLANE, VXI SERVANT

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>degree</b>	IN	VI_NULL (0).
<b>jobId</b>	IN	Specifies an operation identifier.

## Return Values

Completion Codes	Description
VI_SUCCESS	Request serviced successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given object reference is invalid.
VI_ERROR_INV_JOB_ID	Specified job identifier is invalid.
VI_ERROR_INV_DEGREE	Specified <b>degree</b> is invalid.

## Description

This operation is used to request a session to terminate normal execution of an operation, as specified by the **jobId** parameter. The **jobId** parameter is a unique value generated from each call to an asynchronous operation.

If a user passes `VI_NULL` as the **jobId** value to `viTerminate()`, VISA will abort any calls in the current process executing on the specified **vi**. Any call that is terminated this way should return `VI_ERROR_ABORT`. Due to the nature of multi-threaded systems, for example where operations in other threads may complete normally before the operation `viTerminate()` has any effect, the specified return value is not guaranteed.

---

## Related Topics

[VI\\_EVENT\\_IO\\_COMPLETION](#)

[viMoveAsync/viMoveAsyncEx](#)

[viReadAsync](#)

[VISA Resource Template](#)

[viWriteAsync](#)

# viUninstallHandler

## Purpose

Uninstalls handlers for events.

## C Syntax

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

## Visual Basic Syntax

N/A

## Resource Classes

All I/O session types

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>eventType</b>	IN	Logical event identifier.

<b>handler</b>	IN	Interpreted as a valid reference to a handler to be uninstalled by a client application.
<b>userHandle</b>	IN	A value specified by an application that can be used for identifying handlers uniquely in a session for an event.

## Return Values

Completion Codes	Description
VI_SUCCESS	Event handler successfully uninstalled.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
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.

## Description

The `viUninstallHandler()` operation allows 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. Specifying `VI_ANY_HNDLR` as the value for the **handler** parameter causes the operation to uninstall all the handlers with the matching value in the **userHandle** parameter.



**Note** Calling `viUninstallHandler()` removes the specified handler from the list of active handlers on the given session. If no handlers remain for the specified event type, the VISA driver disables that event type on the given session. It is not valid for a user to call this operation from within a callback, because this may cause a deadlock condition within the VISA driver.

## Related Topics

[viDisableEvent](#)

[viEventHandler](#)

[viInstallHandler](#)

[VISA Resource Template](#)

## viUnlock

### Purpose

Relinquishes a lock for the specified resource.

### C Syntax

```
ViStatus viUnlock(ViSession vi)
```

### Visual Basic Syntax

```
viUnlock&(ByVal vi&)
```

## Resource Classes

All I/O session types

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.

## Return Values

Completion Codes	Description
VI_SUCCESS	Lock successfully relinquished.
VI_SUCCESS_NESTED_EXCLUSIVE	Call succeeded, but this session still has nested exclusive locks.
VI_SUCCESS_NESTED_SHARED	Call succeeded, but this session still has nested shared locks.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_SESN_NLOCKED	The current session did not have any lock on the resource.

# Description

This operation is used to relinquish the lock previously obtained using the `viLock()` operation.

---

## Related Topics

[viLock](#)

[VISA Resource Template](#)

# viUnmapAddress

## Purpose

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

## C Syntax

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

## Visual Basic Syntax

```
viUnmapAddress& (ByVal vi&)
```

## Resource Classes

PXI INSTR, PXI MEMACC, VXI INSTR, VXI MEMACC

## Parameters

Name	Direction	Description
vi	IN	Unique logical identifier to a session.

## Return Values

Completion Codes	Description
VI_SUCCESS	Operation completed successfully.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session 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.

## Description

The `viUnmapAddress()` operation unmaps the region previously mapped by the `viMapAddress()` or `viMapAddressEx()` operation for this session.

---

### Related Topics

[INSTR Resource](#)

[MEMACC Resource](#)

[viMapAddress/viMapAddressEx](#)

## viUnmapTrigger



## Purpose

Undo a previous map from the specified trigger source line to the specified destination line.

## C Syntax

```
ViStatus viUnmapTrigger(ViSession vi, ViInt16 trigSrc, ViInt16 trigDest)
```

## Visual Basic Syntax

```
viUnmapTrigger& (ByVal vi&, ByVal trigSrc%, ByVal trigDest%)
```

## Resource Classes

PXI BACKPLANE, VXI BACKPLANE

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>trigSrc</b>	IN	Source line used in previous map. Refer to the <b>Description</b> section for actual values.
<b>trigDest</b>	IN	Destination line used in previous map. Refer to the <b>Description</b> section for actual values.

## Return Values

Completion Codes	Description
------------------	-------------

VI_SUCCESS	The operation completed successfully and the END indicator was received (for interfaces that have END indicators).
------------	--

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given <b>vi</b> does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by <b>vi</b> has been locked for this kind of access.
VI_ERROR_INV_LINE	One of the specified lines ( <b>trigSrc</b> or <b>trigDest</b> ) is invalid.
VI_ERROR_TRIG_NMAPPED	The path from <b>trigSrc</b> to <b>trigDest</b> is not currently mapped.
VI_ERROR_NSUP_LINE	One of the specified lines ( <b>trigSrc</b> or <b>trigDest</b> ) is not supported by this VISA implementation.

## Description

This operation can be used to undo a previous mapping of one trigger line to another. This operation is valid only on BACKPLANE (mainframe) sessions.

## Special Values for trigSrc Parameter

Value	Action Description
VI_TRIG_TTL0 - VI_TRIG_TTL7	Unmap the specified VXI TTL trigger line.
VI_TRIG_ECL0 - VI_TRIG_ECL1	Unmap the specified VXI ECL trigger line.
VI_TRIG_PANEL_IN	Unmap the controller's front panel trigger input line.
VI_TRIG_PANEL_OUT	Unmap the controller's front panel trigger output line.

## Special Values for trigDest Parameter

Value	Action Description
VI_TRIG_TTL0 - VI_TRIG_TTL7	Unmap the specified VXI TTL trigger line.
VI_TRIG_ECL0 - VI_TRIG_ECL1	Unmap the specified VXI ECL trigger line.
VI_TRIG_PANEL_IN	Unmap the controller's front panel trigger input line.
VI_TRIG_PANEL_OUT	Unmap the controller's front panel trigger output line.
VI_TRIG_ALL	Unmap all trigger lines to which <b>trigSrc</b> is currently connected.

This operation unmaps only one trigger mapping per call. In other words, if `viMapTrigger()` was called multiple times on the same BACKPLANE Resource and created multiple mappings for either **trigSrc** or **trigDest**, trigger mappings other than the one

specified by **trigSrc** and **trigDest** should remain in effect after this call completes.

---

## Related Topics

[BACKPLANE Resource](#)

[viMapTrigger](#)

# viUsbControlIn

## Purpose

Performs a USB control pipe transfer from the device.



**Note** This operation is intended only for users familiar with the USB protocol.

## C Syntax

```
ViStatus viUsbControlIn (ViSession vi, ViInt16 bmRequestType,
ViInt16 bRequest, ViUInt16 wValue, ViUInt16 wIndex, ViUInt16 w
Length, ViPBuf buf, ViPUInt16 retCnt);
```

## Visual Basic Syntax

```
viUsbControlIn& (ByVal vi&, ByVal bmRequestType%, ByVal bReque
st%, ByVal wValue%, ByVal wIndex%, ByVal wLength%, buf As Byte, r
etCnt%)
```

## Resource Classes

USB INSTR, USB RAW

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>bmRequestType</b>	IN	The <b>bmRequestType</b> parameter of the setup stage of a USB control transfer. Refer to the USB specification for further details.
<b>bRequest</b>	IN	The <b>bRequest</b> parameter of the setup stage of a USB control transfer.
<b>wValue</b>	IN	The <b>wValue</b> parameter of the setup stage of a USB control transfer.
<b>wIndex</b>	IN	The <b>wIndex</b> parameter of the setup stage of a USB control transfer. This is usually the index of the interface or endpoint.
<b>wLength</b>	IN	The <b>wLength</b> parameter of the setup stage of a USB control transfer. This value also specifies the size of the data buffer to receive the data from the optional data stage of the control transfer.
<b>buf</b>	OUT	The data buffer that receives the data from the optional data stage of the control transfer. This is ignored if <b>wLength</b> is 0.
<b>retCnt</b>	OUT	Number of bytes actually transferred in the optional data stage of the control transfer. This parameter may be <code>VI_NULL</code> if you do not need this information.

## Return Values

Error Codes	Description
<code>VI_ERROR_INV_OBJECT</code>	The given session reference is invalid.
<code>VI_ERROR_RSRC_LOCKED</code>	Specified operation could not be performed because the resource identified by <code>vi</code> has been locked for this kind of access.
<code>VI_ERROR_TMO</code>	Timeout expired before operation completed.
<code>VI_ERROR_INV_SETUP</code>	Unable to start write operation because setup is invalid (due to attributes being set to an inconsistent state).
<code>VI_ERROR_IO</code>	An unknown I/O error occurred during transfer.
<code>VI_ERROR_CONN_LOST</code>	The I/O connection for the given session has been lost.
<code>VI_ERROR_INV_PARAMETER</code>	The value of some parameter—which parameter is not known—is invalid.
<code>VI_ERROR_INV_MASK</code>	The <code>bmRequestType</code> parameter contains an invalid mask.

## Description

The `viUsbControlIn()` operation synchronously performs a USB control pipe

transfer from the device. The values of the data payload in the setup stage of the control transfer are taken as parameters and include **bmRequestType**, **bRequest**, **wValue**, **wIndex**, and **wLength**. An optional data buffer **buf** receives data if a data stage is required for this transfer. Only one USB control pipe transfer operation can occur at any one time.

---

## Related Topics

[INSTR Resource](#)

[RAW Resource](#)

[VI\\_ATTR\\_USB\\_CTRL\\_PIPE](#)

[viUsbControlOut](#)

# viUsbControlOut

## Purpose

Performs a USB control pipe transfer to the device.



**Note** This operation is intended only for users familiar with the USB protocol.

## C Syntax

```
ViStatus viUsbControlOut (ViSession vi, ViInt16 bmRequestType,
ViInt16 bRequest, ViUInt16 wValue, ViUInt16 wIndex, ViUInt16 w
Length, ViPBuf buf) ;
```

## Visual Basic Syntax

```
viUsbControlOut& (ByVal vi&, ByVal bmRequestType%, ByVal bRequ
est%, ByVal wValue%, ByVal wIndex%, ByVal wLength%, buf As Byte)
```

## Resource Classes

USB INSTR, USB RAW

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>bmRequestType</b>	IN	The <b>bmRequestType</b> parameter of the setup stage of a USB control transfer. Refer to the USB specification for further details.
<b>bRequest</b>	IN	The <b>bRequest</b> parameter of the setup stage of a USB control transfer.
<b>wValue</b>	IN	The <b>wValue</b> parameter of the setup stage of a USB control transfer.
<b>wIndex</b>	IN	The <b>wIndex</b> parameter of the setup stage of a USB control transfer. This is usually the index of the interface or endpoint.
<b>wLength</b>	IN	The <b>wLength</b> parameter of the setup stage of a USB control transfer. This value also specifies the size of the data buffer that contains the data to send in the optional data stage of the control transfer.
<b>buf</b>	IN	The data buffer that sends the data in the optional data stage of the control transfer. This is ignored if <b>wLength</b> is 0.



## Return Values

Error Codes	Description
<code>VI_ERROR_INV_OBJECT</code>	The given session reference is invalid.
<code>VI_ERROR_RSRC_LOCKED</code>	Specified operation could not be performed because the resource identified by <code>vi</code> has been locked for this kind of access.
<code>VI_ERROR_TMO</code>	Timeout expired before operation completed.
<code>VI_ERROR_INV_SETUP</code>	Unable to start write operation because setup is invalid (due to attributes being set to an inconsistent state).
<code>VI_ERROR_IO</code>	An unknown I/O error occurred during transfer.
<code>VI_ERROR_CONN_LOST</code>	The I/O connection for the given session has been lost.
<code>VI_ERROR_INV_PARAMETER</code>	The value of some parameter—which parameter is not known—is invalid.
<code>VI_ERROR_INV_MASK</code>	The <code>bmRequestType</code> parameter contains an invalid mask.

## Description

The `viUsbControlOut()` operation synchronously performs a USB control pipe

transfer to the device. The values of the data payload in the setup stage of the control transfer are taken as parameters and include **bmRequestType**, **bRequest**, **wValue**, **wIndex**, and **wLength**. An optional data buffer **buf** contains the data to send if a data stage is required for this transfer. Only one USB control pipe transfer operation can occur at any one time.

---

## Related Topics

[INSTR Resource](#)

[RAW Resource](#)

[VI\\_ATTR\\_USB\\_CTRL\\_PIPE](#)

[viUsbControlIn](#)

## viVPrintf

### Purpose

Converts, formats, and sends the parameters designated by **params** to the device or interface as specified by the format string.

### C Syntax

```
ViStatus viVPrintf(ViSession vi, ViString writeFmt, ViVAList p  
arams)
```

### Visual Basic Syntax

```
viVPrintf&(ByVal vi&, ByVal writeFmt$, params as Any)
```

### Resource Classes

GPIO INSTR, GPIO INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB

## RAW, VXI INSTR, VXI SERVANT

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>writeFmt</b>	IN	String describing the format to apply to <b>params</b> .
<b>params</b>	IN	A list containing the variable number of parameters on which the format string is applied. The formatted data is written to the specified device.

## Return Values

Completion Codes	Description
VI_SUCCESS	Parameters were successfully formatted.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by <b>vi</b> has been locked for this kind of access.
VI_ERROR_IO	Could not perform write operation because of I/O error.

VI_ERROR_TMO	Timeout expired before write operation completed.
VI_ERROR_INV_FMT	A format specifier in the <b>writeFmt</b> string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the <b>writeFmt</b> string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

## Description

This operation is similar to `viPrintf()`, except that the **params** parameters list provides the parameters rather than separate **arg** parameters.

---

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[viPrintf](#)

[viSPrintf](#)

[viVScanf](#)

[viVSPrintf](#)

# viVQueryf

## Purpose

Performs a formatted write and read through a single call to an operation.

## C Syntax

```
ViStatus viVQueryf(ViSession vi, ViString writeFmt, ViString readFmt, ViVAList params)
```

## Visual Basic Syntax

```
viVQueryf&(ByVal vi&, ByVal writeFmt$, ByVal readFmt$, params as Any)
```

## Resource Classes

GPIB INSTR, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, VXI INSTR, VXI SERVANT

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>writeFmt</b>	IN	String describing the format of write arguments.
<b>readFmt</b>	IN	String describing the format of read arguments.
<b>params</b>	IN/OUT	A list containing the variable number of write and read parameters. The write parameters are formatted and written to the specified device. The read parameters store the data read

		from the device after the format string is applied to the data.
--	--	---

Return Values

Completion Codes	Description
VI_SUCCESS	Successfully completed the query operation.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by <code>vi</code> has been locked for this kind of access.
VI_ERROR_IO	Could not perform Read/Write operation because of I/O error.
VI_ERROR_TMO	Timeout occurred before Read/Write operation completed.
VI_ERROR_INV_FMT	A format specifier in the <code>writeFmt</code> or <code>readFmt</code> string is invalid.
VI_ERROR_NSUP_FMT	The format specifier is not supported for current argument type.

VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.
----------------	---

## Description

This operation is similar to `viQueryf()`, except that the **params** parameters list provides the parameters rather than the separate **arg** parameter list



**Note** Because the prototype for this function cannot provide complete type-checking, remember that all output parameters must be passed by reference.

---

## Related Topics

[INSTR Resource](#)

[SOCKET Resource](#)

[viQueryf](#)

## viVScanf

### Purpose

Reads, converts, and formats data using the format specifier. Stores the formatted data in the parameters designated by **params**.

### C Syntax

```
ViStatus viVScanf(ViSession vi, ViString readFmt, ViVAlList params)
```

## Visual Basic Syntax

```
viVScanf&(ByVal vi&, ByVal readFmt$, params as Any)
```

## Resource Classes

GPIB INSTR, GPIB INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>readFmt</b>	IN	String describing the format to apply to <b>params</b> .
<b>params</b>	OUT	A list with the variable number of parameters into which the data is read and the format string is applied.

## Return Values

Completion Codes	Description
VI_SUCCESS	Data was successfully read and formatted into <b>params</b> parameter(s).

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.



VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by <b>vi</b> has been locked for this kind of access.
VI_ERROR_IO	Could not perform read operation because of I/O error.
VI_ERROR_TMO	Timeout expired before read operation completed.
VI_ERROR_INV_FMT	A format specifier in the <b>readFmt</b> string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the <b>readFmt</b> string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

## Description

This operation is similar to `viScanf()`, except that the **params** parameters list provides the parameters rather than separate **arg** parameters.



**Note** Because the prototype for this function cannot provide complete type-checking, remember that all output parameters must be passed by reference.

---

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[viScanf](#)

[viSScanf](#)

[viVPrintf](#)

[viVSScanf](#)

# viVSPrintf

## Purpose

Converts, formats, and sends the parameters designated by **params** to a user-specified buffer as specified by the format string.

## C Syntax

```
ViStatus viVSPrintf(ViSession vi, ViPBuf buf, ViString writeFmt, ViVAlList params)
```

## Visual Basic Syntax

```
viVSPrintf&(ByVal vi&, ByVal buf$, ByVal writeFmt$, params as Any)
```

## Resource Classes

GPIB INSTR, GPIB INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

## Parameters

Name	Direction	Description
------	-----------	-------------

<b>vi</b>	IN	Unique logical identifier to a session.
<b>buf</b>	OUT	Buffer where data is to be written.
<b>writeFmt</b>	IN	The format string to apply to parameters in <code>ViVAList</code> .
<b>params</b>	IN	A list containing the variable number of parameters on which the format string is applied. The formatted data is written to the specified <b>buf</b> .

## Return Values

Completion Codes	Description
<code>VI_SUCCESS</code>	Parameters were successfully formatted.

Error Codes	Description
<code>VI_ERROR_INV_OBJECT</code>	The given session reference is invalid.
<code>VI_ERROR_RSRC_LOCKED</code>	Specified operation could not be performed because the resource identified by <b>vi</b> has been locked for this kind of access.
<code>VI_ERROR_INV_FMT</code>	A format specifier in the <b>writeFmt</b> string is invalid.
<code>VI_ERROR_NSUP_FMT</code>	A format specifier in the <b>writeFmt</b> string is not supported.

VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.
----------------	---

## Description

This operation is similar to `viVPrintf()`, except that the output is not written to the device; it is written to the user-specified buffer. This output buffer is NULL terminated.

If this operation outputs an END indicator before all the arguments are satisfied, then the rest of the **writeFmt** string is ignored and the buffer string is still terminated by a NULL.



**Note** The size of the **buf** parameter should be large enough to hold the formatted I/O contents plus the NULL termination character.

---

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[viPrintf](#)

[viSPrintf](#)

[viVPrintf](#)

[viVSScanf](#)

# viVSScanf

## Purpose

Reads, converts, and formats data from a user-specified buffer using the format specifier. Stores the formatted data in the parameters designated by **params**.

## C Syntax

```
ViStatus viVSScanf(ViSession vi, ViBuf buf, ViString readFmt,  
ViVAList params)
```

## Visual Basic Syntax

```
viVSScanf&(ByVal vi&, ByVal buf$, ByVal readFmt$, params as Any)
```

## Resource Classes

GPIO INSTR, GPIB INTRFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>buf</b>	IN	Buffer from which data is read and formatted.
<b>readFmt</b>	IN	String describing the format to apply to <b>params</b> .
<b>params</b>	OUT	A list with the variable number of parameters into which the data is read and the format string is applied.

## Return Values

Completion Codes	Description
VI_SUCCESS	Data was successfully read and formatted into <b>params</b> parameter(s).

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by <b>vi</b> has been locked for this kind of access.
VI_ERROR_INV_FMT	A format specifier in the <b>readFmt</b> string is invalid.
VI_ERROR_NSUP_FMT	A format specifier in the <b>readFmt</b> string is not supported.
VI_ERROR_ALLOC	The system could not allocate a formatted I/O buffer because of insufficient resources.

## Description

The `viVSScanf()` operation is similar to `viVScanf()`, except that the data is read from a user-specified buffer rather than a device.



**Note** Because the prototype for this function cannot provide complete type checking, remember that all output parameters must be passed by reference.

---

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[viScanf](#)

[viSScanf](#)

[viVScanf](#)

[viVSPrintf](#)

## viVxiCommandQuery

### Purpose

Sends the device a miscellaneous command or query and/or retrieves the response to a previous query.

### C Syntax

```
ViStatus viVxiCommandQuery(ViSession vi, ViUInt16 mode, ViUInt32 cmd, ViPUInt32 response)
```

### Visual Basic Syntax

```
viVxiCommandQuery&(ByVal vi&, ByVal mode%, ByVal cmd&, response&)
```

## Resource Classes

### VXI INSTR

### Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>mode</b>	IN	Specifies whether to issue a command and/or retrieve a response. Refer to the <b>Description</b> section for actual values.
<b>cmd</b>	IN	The miscellaneous command to send.
<b>response</b>	OUT	The response retrieved from the device. If the mode specifies to send a command rather than retrieve a response, you can use <code>VI_NULL</code> for this parameter.

### Return Values

Completion Codes	Description
<code>VI_SUCCESS</code>	The operation completed successfully.

Error Codes	Description
<code>VI_ERROR_INV_OBJECT</code>	The given session reference is invalid.



VI_ERROR_NSUP_OPER	The given <b>vi</b> does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by <b>vi</b> has been locked for this kind of access.
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_INP_PROT_VIOL	Device reported an input protocol error during transfer.
VI_ERROR_BERR	Bus error occurred during transfer.
VI_ERROR_RESP_PENDING	A previous response is still pending, causing a multiple query error.
VI_ERROR_INV_MODE	The value specified by the <b>mode</b> parameter is invalid.

## Description

The `viVxiCommandQuery()` operation can send a command or query, or receive a response to a query previously sent to the device. The **mode** parameter specifies whether to issue a command and/or retrieve a response, and indicates the type or size

of command and/or response to use. The following table defines the values for the **mode** parameter.

Mode	Action Description
VI_VXI_CMD16	Send 16-bit Word Serial command.
VI_VXI_CMD16_RESP16	Send 16-bit Word Serial query; get 16-bit response.
VI_VXI_RESP16	Get 16-bit response from previous query.
VI_VXI_CMD32	Send 32-bit Word Serial command.
VI_VXI_CMD32_RESP16	Send 32-bit Word Serial query; get 16-bit response.
VI_VXI_CMD32_RESP32	Send 32-bit Word Serial query; get 32-bit response.
VI_VXI_RESP32	Get 32-bit response from previous query.

Notice that the **mode** you specify can cause all or part of the **cmd** or **response** parameters to be ignored.

- If **mode** specifies sending a 16-bit command, the upper half of **cmd** is ignored.
- If **mode** specifies retrieving a response only, **cmd** is ignored.
- If **mode** specifies sending a command only, **response** is ignored. You can use `VI_NULL` for the value of **response**.
- If **mode** specifies to retrieve a 16-bit value, the upper half of **response** is set to 0.

Refer to the ***VXI Specification*** for defined Word Serial commands.

## Related Topics

### [INSTR Resource](#)

# viWaitOnEvent

## Purpose

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

## C Syntax

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

## Visual Basic Syntax

```
viWaitOnEvent&(ByVal vi&, ByVal inEventType&, ByVal timeout&, ou  
tEventType&, outContext&)
```

## Resource Classes

All I/O session types

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>inEventType</b>	IN	Logical identifier of the event(s) to wait for.
<b>timeout</b>	IN	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.
<b>outEventType</b>	OUT	Logical identifier of the event actually received.
<b>outContext</b>	OUT	A handle specifying the unique occurrence of an event.

## Return Values

Completion Codes	Description
VI_SUCCESS	Wait terminated successfully on receipt of an event occurrence. The queue is empty.
VI_SUCCESS_QUEUE_EMPTY	Wait terminated successfully on receipt of an event notification. There is still at least one more event occurrence of the type specified by <b>inEventType</b> available for this session.
VI_WARN_QUEUE_OVERFLOW	The event returned is valid. One or more events that occurred have not been raised because there was no room available on the queue at the time of their occurrence. This could happen because <b>VI_ATTR_MAX_QUEUE_LENGTH</b> is not set to a large enough value for your application and/or events are coming in faster than you are servicing them.

Error Codes	Description
-------------	-------------

<code>VI_ERROR_INV_OBJECT</code>	The given session reference is invalid.
<code>VI_ERROR_INV_EVENT</code>	Specified event type is not supported by the resource.
<code>VI_ERROR_TMO</code>	Specified event did not occur within the specified time period.
<code>VI_ERROR_NENABLED</code>	The session must be enabled for events of the specified type in order to receive them.
<code>VI_ERROR_QUEUE_OVERFLOW</code>	No new event is raised because there is no room available on the queue. This means you have already received all previous events but not closed them. You must call <code>viClose</code> on each event you receive from <code>viWaitOnEvent</code> .

## Description

The `viWaitOnEvent()` operation suspends the execution of a thread of an application and waits for an event of the type specified by **inEventType** for a time period specified by **timeout**. You can wait only for events that have been enabled with the `viEnableEvent()` operation. 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 is not suspended; therefore, this value can be used to dequeue events from an event queue.

When the **outContext** handle returned from a successful invocation of `viWaitOnEvent()` is no longer needed, it should be passed to `viClose()`.

If a session's event queue becomes full and a new event arrives, the new event is discarded. The default event queue size (per session) is 50, which is sufficiently large

for most applications. If an application expects more than 50 events to arrive without having been handled, it can modify the value of the attribute `VI_ATTR_MAX_QUEUE_LENGTH` to the required size.

The **outEventType** and **outContext** parameters are optional and can be `VI_NULL`. This can be used if the event type is known from the **inEventType** parameter, or if the **outContext** handle is not needed to retrieve additional information. If `VI_NULL` is used for the **outContext** parameter, VISA will automatically close the event context.

---

## Related Topics

[Events](#)

[VI\\_ATTR\\_MAX\\_QUEUE\\_LENGTH](#)

[viClose](#)

[viEnableEvent](#)

[VISA Resource Template](#)

## viWrite

### Purpose

Writes data to device or interface synchronously.

### C Syntax

```
ViStatus viWrite(ViSession vi, ViBuf buf, ViUInt32 count, ViPU
Int32 retCount)
```

### Visual Basic Syntax

```
viWrite&(ByVal vi&, ByVal buf$, ByVal count&, retCount&)
```

## Resource Classes

GPIO INSTR, GPIO INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>buf</b>	IN	Location of a data block to be sent to a device.
<b>count</b>	IN	Number of bytes to be written.
<b>retCount</b>	OUT	Number of bytes actually transferred.

## Return Values

Completion Codes	Description
VI_SUCCESS	Transfer completed.

Error Codes	Description
VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given vi does not support this operation.

<code>VI_ERROR_RSRC_LOCKED</code>	Specified operation could not be performed because the resource identified by <code>vi</code> has been locked for this kind of access.
<code>VI_ERROR_TMO</code>	Timeout expired before operation completed.
<code>VI_ERROR_RAW_WR_PROT_VIOL</code>	Violation of raw write protocol occurred during transfer.
<code>VI_ERROR_RAW_RD_PROT_VIOL</code>	Violation of raw read protocol occurred during transfer.
<code>VI_ERROR_INP_PROT_VIOL</code>	Device reported an input protocol error during transfer.
<code>VI_ERROR_BERR</code>	Bus error occurred during transfer.
<code>VI_ERROR_INV_SETUP</code>	Unable to start write operation because setup is invalid (due to attributes being set to an inconsistent state).
<code>VI_ERROR_NCIC</code>	The interface associated with the given <code>vi</code> is not currently the controller in charge.
<code>VI_ERROR_NLISTENERS</code>	No-listeners condition is detected (both <code>NRFD</code> and <code>NDAC</code> are unasserted).
<code>VI_ERROR_IO</code>	An unknown I/O error occurred during transfer.
<code>VI_ERROR_CONN_LOST</code>	The I/O connection for the given session has been lost.



## Description

The `viWrite()` 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.



**Note** The **retCount** parameter always is valid on both success and error.

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[viBufWrite](#)

[viRead](#)

[viWriteAsync](#)

[viWriteFromFile](#)

## viWriteAsync

### Purpose

Writes data to device or interface asynchronously.

### C Syntax

```
ViStatus viWriteAsync(ViSession vi, ViBuf buf, ViUInt32 count,
```

ViPJobId **jobId**)

Visual Basic Syntax

N/A

Resource Classes

GPIB INSTR, GPIB INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>buf</b>	IN	Location of a data block to be sent to a device.
<b>count</b>	IN	Number of bytes to be written.
<b>jobId</b>	OUT	Job ID of this asynchronous write operation.

Return Values

Completion Codes	Description
VI_SUCCESS	Asynchronous write operation successfully queued.
VI_SUCCESS_SYNC	Write operation performed synchronously.

Error Codes	Description
<code>VI_ERROR_INV_OBJECT</code>	The given session reference is invalid.
<code>VI_ERROR_RSRC_LOCKED</code>	Specified operation could not be performed because the resource identified by <code>vi</code> has been locked for this kind of access.
<code>VI_ERROR_QUEUE_ERROR</code>	Unable to queue write operation (usually due to the I/O completion event not being enabled or insufficient space in the session's queue).
<code>VI_ERROR_IN_PROGRESS</code>	Unable to queue the asynchronous operation because there is already an operation in progress.

## Description

The `viWriteAsync()` operation asynchronously transfers data. The data to be written is in the buffer represented by **buf**. This operation normally returns before the transfer terminates.

Before calling this operation, you should enable the session for receiving I/O completion events. After the transfer has completed, an I/O completion event is posted.

The operation returns a job identifier that you can use with either `viTerminate()` to abort the operation or with an I/O completion event to identify which asynchronous write operation completed. VISA will never return `VI_NULL` for a valid **jobId**.



**Note** If you have enabled `VI_EVENT_IO_COMPLETION` for queueing (`VI_QUEUE`), for each successful call to `viWriteAsync()`, you must call `viWaitOnEvent()` to retrieve the I/O completion event. This is true even if the I/O is done synchronously (that is, if the operation returns `VI_SUCCESS_SYNC`). If you are using LabVIEW, this is done for you automatically.

---

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[VI\\_EVENT\\_IO\\_COMPLETION](#)

[viEnableEvent](#)

[viReadAsync](#)

[viTerminate](#)

[viWaitOnEvent](#)

[viWrite](#)

## viWriteFromFile

### Purpose

Take data from a file and write it out synchronously.

### C Syntax

```
ViStatus viWriteFromFile(ViSession vi, ViString fileName, ViUInt32 count,  
ViPUInt32 retCount)
```

## Visual Basic Syntax

```
viWriteFromFile& (ByVal vi&, ByVal filename$, ByVal count&, retCount&)
```

## Resource Classes

GPIO INSTR, GPIO INTFC, Serial INSTR, TCPIP INSTR, TCPIP SOCKET, USB INSTR, USB RAW, VXI INSTR, VXI SERVANT

## Parameters

Name	Direction	Description
<b>vi</b>	IN	Unique logical identifier to a session.
<b>fileName</b>	IN	Name of file from which data will be read.
<b>count</b>	IN	Number of bytes to be written.
<b>retCount</b>	OUT	Number of bytes actually transferred.

## Return Values

Completion Codes	Description
VI_SUCCESS	Transfer completed.

Error Codes	Description
-------------	-------------

VI_ERROR_INV_OBJECT	The given session reference is invalid.
VI_ERROR_NSUP_OPER	The given vi does not support this operation.
VI_ERROR_RSRC_LOCKED	Specified operation could not be performed because the resource identified by vi has been locked for this kind of access.
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 unasserted).
VI_ERROR_IO	An unknown I/O error occurred during transfer.

<code>VI_ERROR_FILE_ACCESS</code>	An error occurred while trying to open the specified file. Possible reasons include an invalid path or lack of access rights.
<code>VI_ERROR_FILE_IO</code>	An error occurred while accessing the specified file.
<code>VI_ERROR_CONN_LOST</code>	The I/O connection for the given session has been lost.

## Description

This write operation synchronously transfers data. The file specified in **fileName** is opened in binary read-only mode, and the data (up to end-of-file or the number of bytes specified in count) is read. The data is then written to the device. This operation returns only when the transfer terminates.

This operation is useful for sending data that was already processed and/or formatted.

### Special Values for retCount Parameter

Value	Action Description
<code>VI_NULL</code>	Do not return the number of bytes transferred.

If you pass `VI_NULL` as the **retCount** parameter to the `viWriteFromFile()` operation, the number of bytes transferred will not be returned. This may be useful if it is important to know only whether the operation succeeded or failed.



**Note** The **retCount** parameter always is valid on both success and error.

## Related Topics

[INSTR Resource](#)

[INTFC Resource](#)

[SERVANT Resource](#)

[SOCKET Resource](#)

[viReadToFile](#)

[viWrite](#)



# Resources

This section lists the attributes, events, and operations in each resource in VISA.

Refer to [Attributes](#), [Events](#), or [Operations](#) for more detailed information.

## VISA Resource Template

This section lists the attributes, events, and operations for the VISA Resource Template. The attributes, events, and operations in the VISA Resource Template are available to all other resources.

### Attributes

VI\_ATTR\_MAX\_QUEUE\_LENGTH

VI\_ATTR\_RM\_SESSION

VI\_ATTR\_RSRC\_CLASS

VI\_ATTR\_RSRC\_IMPL\_VERSION

VI\_ATTR\_RSRC\_LOCK\_STATE

VI\_ATTR\_RSRC\_MANF\_ID

VI\_ATTR\_RSRC\_MANF\_NAME

VI\_ATTR\_RSRC\_NAME

VI\_ATTR\_RSRC\_SPEC\_VERSION

VI\_ATTR\_USER\_DATA/VI\_ATTR\_USER\_DATA\_32/VI\_ATTR\_USER\_DATA\_64

## Events

### VI\_EVENT\_EXCEPTION

## Operations

viClose (vi)

viDisableEvent (vi, eventType, mechanism)

viDiscardEvents (vi, eventType, mechanism)

viEnableEvent (vi, eventType, mechanism, context)

viGetAttribute (vi, attribute, attrState)

viInstallHandler (vi, eventType, handler, userHandle)

viLock (vi, lockType, timeout, requestedKey, accessKey)

viSetAttribute (vi, attribute, attrState)

viStatusDesc (vi, status, desc)

viTerminate (vi, degree, jobId)

viUninstallHandler (vi, eventType, handler, userHandle)

viUnlock (vi)

viWaitOnEvent (vi, inEventType, timeout, outEventType, outContext)

## VISA Resource Manager

This section lists the attributes, events, and operations for the VISA Resource Manager. The attributes, events, and operations in the VISA Resource Template are available to

this resource in addition to the operations listed below.

## Attributes

The attributes for the VISA Resource Template are available to this resource. This resource has no defined attributes of its own.

## Events

None

## Operations

viFindNext (findList, instrDesc)

viFindRsrc (sesn, expr, findList, retcnt, instrDesc)

viOpen (sesn, rsrcName, accessMode, timeout, vi)

viOpenDefaultRM (sesn)

viParseRsrc (sesn, rsrcName, intfType, intfNum)

viParseRsrcEx (sesn, rsrcName, intfType, intfNum, rsrcClass, unaliasdExpandedRsrcName, aliasIfExists)

## INSTR Resource

This section lists the attributes, events, and operations for the INSTR Resource. The attributes, events, and operations in the VISA Resource Template are available to this resource in addition to the attributes and operations listed below.

## Attributes

VI\_ATTR\_ASRL\_ALLOW\_TRANSMIT

VI\_ATTR\_ASRL\_AVAIL\_NUM

VI\_ATTR\_ASRL\_BAUD

VI\_ATTR\_ASRL\_BREAK\_LEN

VI\_ATTR\_ASRL\_BREAK\_STATE

VI\_ATTR\_ASRL\_CTS\_STATE

VI\_ATTR\_ASRL\_DATA\_BITS

VI\_ATTR\_ASRL\_DCD\_STATE

VI\_ATTR\_ASRL\_DISCARD\_NULL

VI\_ATTR\_ASRL\_DSR\_STATE

VI\_ATTR\_ASRL\_DTR\_STATE

VI\_ATTR\_ASRL\_END\_IN

VI\_ATTR\_ASRL\_END\_OUT

VI\_ATTR\_ASRL\_FLOW\_CONTROL

VI\_ATTR\_ASRL\_PARITY

VI\_ATTR\_ASRL\_REPLACE\_CHAR

VI\_ATTR\_ASRL\_RI\_STATE

VI\_ATTR\_ASRL\_RTS\_STATE

VI\_ATTR\_ASRL\_STOP\_BITS

VI\_ATTR\_ASRL\_WIRE\_MODE

VI\_ATTR\_ASRL\_XOFF\_CHAR

VI\_ATTR\_ASRL\_XON\_CHAR

VI\_ATTR\_CMDR\_LA

VI\_ATTR\_DEST\_ACCESS\_PRIV

VI\_ATTR\_DEST\_BYTE\_ORDER

VI\_ATTR\_DEST\_INCREMENT

VI\_ATTR\_DMA\_ALLOW\_EN

VI\_ATTR\_FDC\_CHNL

VI\_ATTR\_FDC\_MODE

VI\_ATTR\_FDC\_USE\_PAIR

VI\_ATTR\_FILE\_APPEND\_EN

VI\_ATTR\_GPIB\_PRIMARY\_ADDR

VI\_ATTR\_GPIB\_READDR\_EN

VI\_ATTR\_GPIB\_REN\_STATE

VI\_ATTR\_GPIB\_SECONDARY\_ADDR

VI\_ATTR\_GPIB\_UNADDR\_EN

VI\_ATTR\_IMMEDIATE\_SERV

VI\_ATTR\_INTF\_INST\_NAME

VI\_ATTR\_INTF\_NUM

VI\_ATTR\_INTF\_TYPE

VI\_ATTR\_IO\_PROT

VI\_ATTR\_MAINFRAME\_LA

VI\_ATTR\_MANF\_ID

VI\_ATTR\_MANF\_NAME

VI\_ATTR\_MEM\_BASE/VI\_ATTR\_MEM\_BASE\_32/VI\_ATTR\_MEM\_BASE\_64

VI\_ATTR\_MEM\_SIZE/VI\_ATTR\_MEM\_SIZE\_32/VI\_ATTR\_MEM\_SIZE\_64

VI\_ATTR\_MEM\_SPACE

VI\_ATTR\_MODEL\_CODE

VI\_ATTR\_MODEL\_NAME

VI\_ATTR\_RD\_BUF\_OPER\_MODE

VI\_ATTR\_SEND\_END\_EN

VI\_ATTR\_SLOT

VI\_ATTR\_SRC\_ACCESS\_PRIV

VI\_ATTR\_SRC\_BYTE\_ORDER

VI\_ATTR\_SRC\_INCREMENT

VI\_ATTR\_SUPPRESS\_END\_EN

VI\_ATTR\_TCPIP\_ADDR

VI\_ATTR\_TCPIP\_DEVICE\_NAME

VI\_ATTR\_TCPIP\_HOSTNAME

VI\_ATTR\_TERMCHAR

VI\_ATTR\_TERMCHAR\_EN

VI\_ATTR\_TMO\_VALUE

VI\_ATTR\_TRIG\_ID

VI\_ATTR\_VXI\_DEV\_CLASS

VI\_ATTR\_VXI\_LA

VI\_ATTR\_VXI\_TRIG\_DIR

VI\_ATTR\_VXI\_TRIG\_LINES\_EN

VI\_ATTR\_VXI\_TRIG\_SUPPORT

VI\_ATTR\_WIN\_ACCESS

VI\_ATTR\_WIN\_ACCESS\_PRIV

VI\_ATTR\_WIN\_BASE\_ADDR/VI\_ATTR\_WIN\_BASE\_ADDR\_32/VI\_ATTR\_WIN\_BASE\_ADDR\_64

VI\_ATTR\_WIN\_BYTE\_ORDER

VI\_ATTR\_WIN\_SIZE/VI\_ATTR\_WIN\_SIZE\_32/VI\_ATTR\_WIN\_SIZE\_64

VI\_ATTR\_WR\_BUF\_OPER\_MODE

## Events

VI\_EVENT\_IO\_COMPLETION

VI\_EVENT\_SERVICE\_REQ

VI\_EVENT\_TRIG

VI\_EVENT\_VXI\_SIGP

VI\_EVENT\_VXI\_VME\_INTR

## Operations

viAssertTrigger (vi, protocol)

viBufRead (vi, buf, count, retCount)

viBufWrite (vi, buf, count, retCount)

viClear (vi)

viFlush (vi, mask)

viGpibControlREN (vi, mode)

viIn8 (vi, space, offset, val8)

viIn8Ex (vi, space, offset, val8)

viIn16 (vi, space, offset, val16)

viIn16Ex (vi, space, offset, val16)

viIn32 (vi, space, offset, val32)

viIn32Ex (vi, space, offset, val32)

viIn64 (vi, space, offset, val64)

viIn64Ex (vi, space, offset, val64)

viMapAddress (vi, mapSpace, mapBase, mapSize, access, suggested, address)

viMapAddressEx (vi, mapSpace, mapBase, mapSize, access, suggested, address)

viMemAlloc (vi, size, offset)

viMemAllocEx (vi, size, offset)



viMemFree (vi, offset)

viMemFreeEx (vi, offset)

viMove (vi, srcSpace, srcOffset, srcWidth, destSpace, destOffset, destWidth, length)

viMoveEx (vi, srcSpace, srcOffset, srcWidth, destSpace, destOffset, destWidth, length)

viMoveAsync (vi, srcSpace, srcOffset, srcWidth, destSpace, destOffset, destWidth, length, jobId)

viMoveAsyncEx (vi, srcSpace, srcOffset, srcWidth, destSpace, destOffset, destWidth, length, jobId)

viMoveIn8 (vi, space, offset, length, buf8)

viMoveIn8Ex (vi, space, offset, length, buf8)

viMoveIn16 (vi, space, offset, length, buf16)

viMoveIn16Ex (vi, space, offset, length, buf16)

viMoveIn32 (vi, space, offset, length, buf32)

viMoveIn32Ex (vi, space, offset, length, buf32)

viMoveIn64 (vi, space, offset, length, buf64)

viMoveIn64Ex (vi, space, offset, length, buf64)

viMoveOut8 (vi, space, offset, length, buf8)

viMoveOut8Ex (vi, space, offset, length, buf8)

viMoveOut16 (vi, space, offset, length, buf16)

viMoveOut16Ex (vi, space, offset, length, buf16)

viMoveOut32 (vi, space, offset, length, buf32)

viMoveOut32Ex (vi, space, offset, length, buf32)

viMoveOut64 (vi, space, offset, length, buf64)

viMoveOut64Ex (vi, space, offset, length, buf64)

viOut8 (vi, space, offset, val8)

viOut8Ex (vi, space, offset, val8)

viOut16 (vi, space, offset, val16)

viOut16Ex (vi, space, offset, val16)

viOut32 (vi, space, offset, val32)

viOut32Ex (vi, space, offset, val32)

viOut64 (vi, space, offset, val64)

viOut64Ex (vi, space, offset, val64)

viPeek8 (vi, addr, val8)

viPeek16 (vi, addr, val16)

viPeek32 (vi, addr, val32)

viPeek64 (vi, addr, val64)

viPoke8 (vi, addr, val8)

viPoke16 (vi, addr, val16)

viPoke32 (vi, addr, val32)

viPoke64 (vi, addr, val64)

viPrintf (vi, writeFmt, ...)

viQueryf (vi, writeFmt, readFmt, ...)

viRead (vi, buf, count, retCount)

viReadAsync (vi, buf, count, jobId)

viReadSTB (vi, status)

viReadToFile (vi, fileName, count, retCount)

viScanf (vi, readFmt, ...)

viSetBuf (vi, mask, size)

viSPrintf (vi, buf, writeFmt, ...)

viSScanf (vi, buf, readFmt, ...)

viUnmapAddress (vi)

viUsbControlIn

viUsbControlOut

viVPrintf (vi, writeFmt, params)

viVQueryf (vi, writeFmt, readFmt, params)

viVScanf (vi, readFmt, params)

viVSPrintf (vi, buf, writeFmt, params)

viVSScanf (vi, buf, readFmt, params)

viVxiCommandQuery (vi, mode, cmd, response)

viWrite (vi, buf, count, retCount)

viWriteAsync(vi, buf, count, jobId)

## MEMACC Resource

This section lists the attributes, events, and operations for the MEMACC Resource. The attributes, events, and operations in the VISA Resource Template are available to this resource in addition to the attributes and operations listed below.

### Attributes

VI\_ATTR\_DEST\_ACCESS\_PRIV

VI\_ATTR\_DEST\_BYTE\_ORDER

VI\_ATTR\_DEST\_INCREMENT

VI\_ATTR\_DMA\_ALLOW\_EN

VI\_ATTR\_GPIB\_PRIMARY\_ADDR

VI\_ATTR\_GPIB\_SECONDARY\_ADDR

VI\_ATTR\_INTF\_INST\_NAME

VI\_ATTR\_INTF\_NUM

VI\_ATTR\_INTF\_TYPE

VI\_ATTR\_SRC\_ACCESS\_PRIV

VI\_ATTR\_SRC\_BYTE\_ORDER

VI\_ATTR\_SRC\_INCREMENT

VI\_ATTR\_TMO\_VALUE

VI\_ATTR\_VXI\_LA

VI\_ATTR\_WIN\_ACCESSVI\_ATTR\_WIN\_ACCESS\_PRIVVI\_ATTR\_WIN\_BASE\_ADDR/VI\_ATTR\_WIN\_BASE\_ADDR\_32/VI\_ATTR\_WIN\_BASE\_ADDR\_64VI\_ATTR\_WIN\_BYTE\_ORDERVI\_ATTR\_WIN\_SIZE/VI\_ATTR\_WIN\_SIZE\_32/VI\_ATTR\_WIN\_SIZE\_64

## Events

VI\_EVENT\_IO\_COMPLETION

## Operations

viIn8 (vi, space, offset, val8)viIn8Ex (vi, space, offset, val8)viIn16 (vi, space, offset, val16)viIn16Ex (vi, space, offset, val16)viIn32 (vi, space, offset, val32)viIn32Ex (vi, space, offset, val32)viIn64 (vi, space, offset, val64)viIn64Ex (vi, space, offset, val64)viMapAddress (vi, mapSpace, mapBase, mapSize, access, suggested, address)viMapAddressEx (vi, mapSpace, mapBase, mapSize, access, suggested, address)

viMemAlloc (vi, size, offset)

viMemAllocEx (vi, size, offset)

viMove (vi, srcSpace, srcOffset, srcWidth, destSpace, destOffset, destWidth, length)

viMoveEx (vi, srcSpace, srcOffset, srcWidth, destSpace, destOffset, destWidth, length)

viMoveAsync (vi, srcSpace, srcOffset, srcWidth, destSpace, destOffset, destWidth, length, jobId)

viMoveAsyncEx (vi, srcSpace, srcOffset, srcWidth, destSpace, destOffset, destWidth, length, jobId)

viMoveIn8 (vi, space, offset, length, buf8)

viMoveIn8Ex (vi, space, offset, length, buf8)

viMoveIn16 (vi, space, offset, length, buf16)

viMoveIn16Ex (vi, space, offset, length, buf16)

viMoveIn32 (vi, space, offset, length, buf32)

viMoveIn32Ex (vi, space, offset, length, buf32)

viMoveIn64 (vi, space, offset, length, buf64)

viMoveIn64Ex (vi, space, offset, length, buf64)

viMoveOut8 (vi, space, offset, length, buf8)

viMoveOut8Ex (vi, space, offset, length, buf8)

viMoveOut16 (vi, space, offset, length, buf16)

viMoveOut16Ex (vi, space, offset, length, buf16)

viMoveOut32 (vi, space, offset, length, buf32)

viMoveOut32Ex (vi, space, offset, length, buf32)

viMoveOut64 (vi, space, offset, length, buf64)

viMoveOut64Ex (vi, space, offset, length, buf64)

viOut8 (vi, space, offset, val8)

viOut8Ex (vi, space, offset, val8)

viOut16 (vi, space, offset, val16)

viOut16Ex (vi, space, offset, val16)

viOut32 (vi, space, offset, val32)

viOut32Ex (vi, space, offset, val32)

viOut64 (vi, space, offset, val64)

viOut64Ex (vi, space, offset, val64)

viPeek8 (vi, addr, val8)

viPeek16 (vi, addr, val16)

viPeek32 (vi, addr, val32)

viPeek64 (vi, addr, val64)

viPoke8 (vi, addr, val8)

viPoke16 (vi, addr, val16)

viPoke32 (vi, addr, val32)

viPoke64 (vi, addr, val64)

viUnmapAddress (vi)

## INTFC Resource

This section lists the attributes, events, and operations for the INTFC Resource. The attributes, events, and operations in the VISA Resource Template are available to this resource in addition to the attributes and operations listed below.

### Attributes

VI\_ATTR\_DEV\_STATUS\_BYTE

VI\_ATTR\_EVENT\_TYPE

VI\_ATTR\_FILE\_APPEND\_EN

VI\_ATTR\_GPIB\_ATN\_STATE

VI\_ATTR\_GPIB\_CIC\_STATE

VI\_ATTR\_GPIB\_HS488\_CBL\_LEN

VI\_ATTR\_GPIB\_NDAC\_STATE

VI\_ATTR\_GPIB\_PRIMARY\_ADDR

VI\_ATTR\_GPIB\_REN\_STATE

VI\_ATTR\_GPIB\_SECONDARY\_ADDR

VI\_ATTR\_GPIB\_SRQ\_STATE

VI\_ATTR\_GPIB\_SYS\_CNTRL\_STATE

VI\_ATTR\_INTF\_INST\_NAME

VI\_ATTR\_INTF\_NUM



VI\_ATTR\_INTF\_TYPE

VI\_ATTR\_MAX\_QUEUE\_LENGTH

VI\_ATTR\_RD\_BUF\_OPER\_MODE

VI\_ATTR\_RM\_SESSION

VI\_ATTR\_RSRC\_IMPL\_VERSION

VI\_ATTR\_RSRC\_LOCK\_STATE

VI\_ATTR\_RSRC\_MANF\_ID

VI\_ATTR\_RSRC\_MANF\_NAME

VI\_ATTR\_RSRC\_NAME

VI\_ATTR\_RSRC\_SPEC\_VERSION

VI\_ATTR\_SEND\_END\_EN

VI\_ATTR\_TERMCHAR

VI\_ATTR\_TERMCHAR\_EN

VI\_ATTR\_TMO\_VALUE

VI\_ATTR\_TRIG\_ID

VI\_ATTR\_USER\_DATA/VI\_ATTR\_USER\_DATA\_32/VI\_ATTR\_USER\_DATA\_64

VI\_ATTR\_WR\_BUF\_OPER\_MODE

## Events

VI\_EVENT\_CLEAR

VI\_EVENT\_GPIB\_CIC

VI\_EVENT GPIB LISTEN

VI\_EVENT GPIB TALK

VI\_EVENT IO COMPLETION

VI\_EVENT SERVICE REQ

VI\_EVENT TRIG

## Operations

ViAssertTrigger (vi, protocol)

viBufRead (vi, buf, count, retCount)

viBufWrite (vi, buf, count, retCount)

viFlush (vi, mask)

viGpibCommand (vi, buf, count, retCount)

viGpibControlATN (vi, mode)

viGpibControlREN (vi, mode)

viGpibPassControl (vi, primAddr, secAddr)

viGpibSendIFC (vi)

viPrintf (vi, writeFmt, ...)

viRead (vi, buf, count, retCount)

viReadAsync (vi, buf, count, jobId)

viReadToFile (vi, fileName, count, retCount)

viScanf (vi, readFmt, ...)

viSetBuf (vi, mask, size)

viSPrintf (vi, buf, writeFmt, ...)

viSScanf (vi, buf, readFmt, ...)

viVPrintf (vi, writeFmt, params)

viVScanf (vi, readFmt, params)

viVSPrintf (vi, buf, writeFmt, params)

viVSScanf (vi, buf, readFmt, params)

viWrite (vi, buf, count, retCount)

viWriteAsync (vi, buf, count, jobId)

viWriteFromFile (vi, fileName, count, retCount)

## BACKPLANE Resource

This section lists the attributes, events, and operations for the BACKPLANE Resource. The attributes, events, and operations in the VISA Resource Template are available to this resource in addition to the attributes and operations listed below.

### Attributes

VI\_ATTR\_GPIB\_PRIMARY\_ADDR

VI\_ATTR\_GPIB\_SECONDARY\_ADDR

VI\_ATTR\_INTF\_INST\_NAME

VI\_ATTR\_INTF\_NUM

VI\_ATTR\_INTF\_TYPE

VI\_ATTR\_MAINFRAME\_LA

VI\_ATTR\_PXI\_CHASSIS

VI\_ATTR\_PXI\_DEST\_TRIG\_BUS

VI\_ATTR\_PXI\_SRC\_TRIG\_BUS

VI\_ATTR\_PXI\_TRIG\_BUS

VI\_ATTR\_TMO\_VALUE

VI\_ATTR\_TRIG\_ID

VI\_ATTR\_VXI\_TRIG\_STATUS

VI\_ATTR\_VXI\_TRIG\_SUPPORT

VI\_ATTR\_VXI\_VME\_INTR\_STATUS

VI\_ATTR\_VXI\_VME\_SYSFAIL\_STATE

## Events

VI\_EVENT\_TRIG

VI\_EVENT\_VXI\_VME\_SYSFAIL

VI\_EVENT\_VXI\_VME\_SYSRESET

## Operations

viAssertIntrSignal (vi, mode, statusID)

viAssertTrigger (vi, protocol)

viAssertUtilSignal (vi, line)

viMapTrigger (vi, trigSrc, trigDest, mode)

viUnmapTrigger (vi, trigSrc, trigDest)

## SERVANT Resource

This section lists the attributes, events, and operations for the SERVANT Resource. The attributes, events, and operations in the VISA Resource Template are available to this resource in addition to the attributes and operations listed below.

### Attributes

VI\_ATTR\_CMDR\_LA

VI\_ATTR\_DEV\_STATUS\_BYTE

VI\_ATTR\_DMA\_ALLOW\_EN

VI\_ATTR\_FILE\_APPEND\_EN

VI\_ATTR\_INTF\_INST\_NAME

VI\_ATTR\_INTF\_NUM

VI\_ATTR\_INTF\_TYPE

VI\_ATTR\_IO\_PROT

VI\_ATTR\_RD\_BUF\_OPER\_MODE

VI\_ATTR\_SEND\_END\_EN

VI\_ATTR\_TERMCHAR

VI\_ATTR\_TERMCHAR\_EN

VI\_ATTR\_TMO\_VALUE

VI\_ATTR\_TRIG\_ID

VI\_ATTR\_VXI\_LA

VI\_ATTR\_VXI\_VME\_SYSFAIL\_STATE

VI\_ATTR\_WR\_BUF\_OPER\_MODE

## Events

VI\_EVENT\_CLEAR

VI\_EVENT\_IO\_COMPLETION

VI\_EVENT\_TRIG

VI\_EVENT\_VXI\_VME\_SYSRESET

## Operations

viAssertIntrSignal (vi, mode, statusID)

viAssertUtilSignal (vi, line)

viBufRead (vi, buf, count, retCount)

viBufWrite (vi, buf, count, retCount)

viFlush (vi, mask)

viPrintf (vi, writeFmt, ...)

viRead (vi, buf, count, retCount)

viReadAsync (vi, buf, count, jobId)

viReadToFile (vi, fileName, count, retCount)

viScanf (vi, readFmt, ...)

viSetBuf (vi, mask, size)

viSprintf (vi, buf, writeFmt, ...)

viSScanf (vi, buf, readFmt, ...)

viVPrintf (vi, writeFmt, params)

viVScanf (vi, readFmt, params)

viVSprintf (vi, buf, writeFmt, params)

viVSScanf (vi, buf, readFmt, params)

viWrite (vi, buf, count, retCount)

viWriteAsync (vi, buf, count, jobId)

viWriteFromFile (vi, fileName, count, retCount)

## SOCKET Resource

This section lists the attributes, events, and operations for the SOCKET Resource. The attributes, events, and operations in the VISA Resource Template are available to this resource in addition to the attributes and operations listed below.

### Attributes

VI\_ATTR\_FILE\_APPEND\_EN

VI\_ATTR\_INTF\_INST\_NAME

VI\_ATTR\_INTF\_NUM

VI\_ATTR\_INTF\_TYPE

VI\_ATTR\_IO\_PROT

VI\_ATTR\_RD\_BUF\_OPER\_MODE

VI\_ATTR\_SEND\_END\_EN

VI\_ATTR\_TCPIP\_ADDR

VI\_ATTR\_TCPIP\_HOSTNAME

VI\_ATTR\_TCPIP\_KEEPA\_LIVE

VI\_ATTR\_TCPIP\_NODELAY

VI\_ATTR\_TCPIP\_PORT

VI\_ATTR\_TERMCHAR

VI\_ATTR\_TERMCHAR\_EN

VI\_ATTR\_TMO\_VALUE

VI\_ATTR\_TRIG\_ID

VI\_ATTR\_WR\_BUF\_OPER\_MODE

## Events

VI\_EVENT\_IO\_COMPLETION

## Operations

viAssertTrigger (vi, protocol)

viBufRead (vi, buf, count, retCount)

viBufWrite (vi, buf, count, retCount)



viClear (vi)

viFlush (vi, mask)

viPrintf (vi, writeFmt, ...)

viQueryf (vi, writeFmt, readFmt, ...)

viRead (vi, buf, count, retCount)

viReadAsync (vi, buf, count, jobId)

viReadSTB (vi, status)

viReadToFile (vi, fileName, count, retCount)

viScanf (vi, readFmt, ...)

viSetBuf (vi, mask, size)

viSPrintf (vi, buf, writeFmt, ...)

viSScanf (vi, buf, readFmt, ...)

viVPrintf (vi, writeFmt, params)

viVQueryf (vi, writeFmt, readFmt, params)

viVScanf (vi, readFmt, params)

viVSPrintf (vi, buf, writeFmt, params)

viVSScanf (vi, buf, readFmt, params)

viWrite (vi, buf, count, retCount)

viWriteAsync (vi, buf, count, jobId)

viWriteFromFile (vi, fileName, count, retCount)

## RAW Resource

This section lists the attributes, events, and operations for the RAW Resource. The attributes, events, and operations in the VISA Resource Template are available to this resource in addition to the attributes and operations listed below.

### Attributes

VI\_ATTR\_FILE\_APPEND\_EN

VI\_ATTR\_INTF\_INST\_NAME

VI\_ATTR\_INTF\_NUM

VI\_ATTR\_INTF\_TYPE

VI\_ATTR\_IO\_PROT

VI\_ATTR\_MANF\_ID

VI\_ATTR\_MANF\_NAME

VI\_ATTR\_MAX\_QUEUE\_LENGTH

VI\_ATTR\_MODEL\_CODE

VI\_ATTR\_MODEL\_NAME

VI\_ATTR\_RD\_BUF\_OPER\_MODE

VI\_ATTR\_RD\_BUF\_SIZE

VI\_ATTR\_RSRC\_CLASS

VI\_ATTR\_RSRC\_IMPL\_VERSION

VI\_ATTR\_RSRC\_LOCK\_STATE

VI\_ATTR\_RSRC\_MANF\_ID

VI\_ATTR\_RSRC\_MANF\_NAME

VI\_ATTR\_RSRC\_NAME

VI\_ATTR\_RSRC\_SPEC\_VERSION

VI\_ATTR\_SUPPRESS\_END\_EN

VI\_ATTR\_TERMCHAR

VI\_ATTR\_TERMCHAR\_EN

VI\_ATTR\_TMO\_VALUE

VI\_ATTR\_USB\_ALT\_SETTING

VI\_ATTR\_USB\_BULK\_IN\_PIPE

VI\_ATTR\_USB\_BULK\_IN\_STATUS

VI\_ATTR\_USB\_BULK\_OUT\_PIPE

VI\_ATTR\_USB\_BULK\_OUT\_STATUS

VI\_ATTR\_USB\_CLASS

VI\_ATTR\_USB\_END\_IN

VI\_ATTR\_USB\_INTFC\_NUM

VI\_ATTR\_USB\_INTR\_IN\_PIPE

VI\_ATTR\_USB\_INTR\_IN\_STATUS

VI\_ATTR\_USB\_MAX\_INTR\_SIZE

VI\_ATTR\_USB\_NUM\_INTFCS

VI\_ATTR\_USB\_NUM\_PIPES

VI\_ATTR\_USB\_PROTOCOL

VI\_ATTR\_USB\_SERIAL\_NUM

VI\_ATTR\_USB\_SUBCLASS

VI\_ATTR\_USER\_DATA/VI\_ATTR\_USER\_DATA\_32/VI\_ATTR\_USER\_DATA\_64

VI\_ATTR\_WR\_BUF\_OPER\_MODE

VI\_ATTR\_WR\_BUF\_SIZE

## Events

VI\_EVENT\_EXCEPTION

VI\_EVENT\_IO\_COMPLETION

VI\_EVENT\_USB\_INTR

## Operations

viAssertTrigger

viBufRead

viBufWrite

viClear

viClose

viDisableEvent

viDiscardEvents

[viEnableEvent](#)

[viEventHandler](#)

[viFindNext](#)

[viFindRsrc](#)

[viGetAttribute](#)

[viInstallHandler](#)

[viLock](#)

[viOpenDefaultRM](#)

[viOpen](#)

[viParseRsrc](#)

[viParseRsrcEx](#)

[viPrintf](#)

[viReadAsync](#)

[viReadSTB](#)

[viReadToFile](#)

[viRead](#)

[viScanf](#)

[viSetAttribute](#)

[viTerminate](#)

[viUninstallHandler](#)

viUnlock

viUsbControlIn

viUsbControlOut

viVPrintf

viVScanf

viVSPrintf

viVSScanf

viUsbControlIn

viUsbControlOut

viWaitOnEvent

viWriteAsync

viWriteFromFile

viWrite

# Error Codes

This topic lists and describes the error codes.

Completion Codes	Values	Meaning
VI_ERROR_SYSTEM_ERROR	BFFF0000h	Unknown system error (miscellaneous error).
VI_ERROR_INV_OBJECT	BFFF000Eh	The given session or object reference is invalid.
VI_ERROR_RSRC_LOCKED	BFFF000Fh	Specified type of lock cannot be obtained or specified operation cannot be performed, because the resource is locked.
VI_ERROR_INV_EXPR	BFFF0010h	Invalid expression specified for search.
VI_ERROR_RSRC_NFOUND	BFFF0011h	Insufficient location information or the device or resource is not present in the system.
VI_ERROR_INV_RSRC_NAME	BFFF0012h	Invalid resource reference specified. Parsing error.
VI_ERROR_INV_ACC_MODE	BFFF0013h	Invalid access mode.
VI_ERROR_TMO	BFFF0015h	Timeout expired before operation completed.
VI_ERROR_CLOSING_FAILED	BFFF0016h	Unable to deallocate the previously allocated data structures corresponding to this session or object reference.
VI_ERROR_INV_DEGREE	BFFF001Bh	Specified degree is invalid.
VI_ERROR_INV_JOB_ID	BFFF001Ch	Specified job identifier is invalid.
VI_ERROR_NSUP_ATTR	BFFF001Dh	The specified attribute is not defined or supported by the referenced session, event, or find list.
VI_ERROR_NSUP_ATTR_STATE	BFFF001Eh	The specified state of the attribute is not valid, or is not supported as defined by the session, event, or find list.

VI_ERROR_ATTR_READONLY	BFFF001Fh	The specified attribute is Read Only.
VI_ERROR_INV_LOCK_TYPE	BFFF0020h	The specified type of lock is not supported by this resource.
VI_ERROR_INV_ACCESS_KEY	BFFF0021h	The access key to the resource associated with this session is invalid.
VI_ERROR_INV_EVENT	BFFF0026h	Specified event type is not supported by the resource.
VI_ERROR_INV_MECH	BFFF0027h	Invalid mechanism specified.
VI_ERROR_HNDLR_NINSTALLED	BFFF0028h	A handler is not currently installed for the specified event.
VI_ERROR_INV_HNDLR_REF	BFFF0029h	The given handler reference is invalid.
VI_ERROR_INV_CONTEXT	BFFF002Ah	Specified event context is invalid.
VI_ERROR_QUEUE_OVERFLOW	BFFF002Dh	The event queue for the specified type has overflowed (usually due to previous events not having been closed).
VI_ERROR_NENABLED	BFFF002Fh	The session must be enabled for events of the specified type in order to receive them.
VI_ERROR_ABORT	BFFF0030h	The operation was aborted.
VI_ERROR_RAW_WR_PROT_VIOL	BFFF0034h	Violation of raw write protocol occurred during transfer.
VI_ERROR_RAW_RD_PROT_VIOL	BFFF0035h	Violation of raw read protocol occurred during transfer.
VI_ERROR_OUTP_PROT_VIOL	BFFF0036h	Device reported an output protocol error during transfer.
VI_ERROR_INP_PROT_VIOL	BFFF0037h	Device reported an input protocol error during transfer.
VI_ERROR_BERR	BFFF0038h	Bus error occurred during transfer.
VI_ERROR_IN_PROGRESS	BFFF0039h	Unable to queue the asynchronous operation because there is already an operation in progress.
VI_ERROR_INV_SETUP	BFFF003Ah	Unable to start operation because setup is invalid (due to attributes being set to an inconsistent state).



VI_ERROR_QUEUE_ERROR	BFFF003Bh	Unable to queue asynchronous operation (usually due to the I/O completion event not being enabled or insufficient space in the session's queue).
VI_ERROR_ALLOC	BFFF003Ch	Insufficient system resources to perform necessary memory allocation.
VI_ERROR_INV_MASK	BFFF003Dh	Invalid buffer mask specified.
VI_ERROR_IO	BFFF003Eh	Could not perform operation because of I/O error.
VI_ERROR_INV_FMT	BFFF003Fh	A format specifier in the format string is invalid.
VI_ERROR_NSUP_FMT	BFFF0041h	A format specifier in the format string is not supported.
VI_ERROR_LINE_IN_USE	BFFF0042h	The specified trigger line is currently in use.
VI_ERROR_NSUP_MODE	BFFF0046h	The specified mode is not supported by this VISA implementation.
VI_ERROR_SRQ_NOCCURRED	BFFF004Ah	Service request has not been received for the session.
VI_ERROR_INV_SPACE	BFFF004Eh	Invalid address space specified.
VI_ERROR_INV_OFFSET	BFFF0051h	Invalid offset specified.
VI_ERROR_INV_WIDTH	BFFF0052h	Invalid source or destination width specified.
VI_ERROR_NSUP_OFFSET	BFFF0054h	Specified offset is not accessible from this hardware.
VI_ERROR_NSUP_VAR_WIDTH	BFFF0055h	Cannot support source and destination widths that are different.
VI_ERROR_WINDOW_NMAPPED	BFFF0057h	The specified session is not currently mapped.
VI_ERROR_RESP_PENDING	BFFF0059h	A previous response is still pending, causing a multiple query error.
VI_ERROR_NLISTENERS	BFFF005Fh	No Listeners condition is detected (both NRF D and NDAC are deasserted).
VI_ERROR_NCIC	BFFF0060h	The interface associated with this session is not currently the controller in charge.

VI_ERROR_NSYS_CNTL	BFFF0061h	The interface associated with this session is not the system controller.
VI_ERROR_NSUP_OPER	BFFF0067h	The given session or object reference does not support this operation.
VI_ERROR_INTR_PENDING	BFFF0068h	An interrupt is still pending from a previous call.
VI_ERROR_ASRL_PARITY	BFFF006Ah	A parity error occurred during transfer.
VI_ERROR_ASRL_FRAMING	BFFF006Bh	A framing error occurred during transfer.
VI_ERROR_ASRL_OVERRUN	BFFF006Ch	An overrun error occurred during transfer. A character was not read from the hardware before the next character arrived.
VI_ERROR_TRIG_NMAPPED	BFFF006Eh	The path from trigSrc to trigDest is not currently mapped.
VI_ERROR_NSUP_ALIGN_OFFSET	BFFF0070h	The specified offset is not properly aligned for the access width of the operation.
VI_ERROR_USER_BUF	BFFF0071h	A specified user buffer is not valid or cannot be accessed for the required size.
VI_ERROR_RSRC_BUSY	BFFF0072h	The resource is valid, but VISA cannot currently access it.
VI_ERROR_NSUP_WIDTH	BFFF0076h	Specified width is not supported by this hardware.
VI_ERROR_INV_PARAMETER	BFFF0078h	The value of some parameter—which parameter is not known—is invalid.
VI_ERROR_INV_PROT	BFFF0079h	The protocol specified is invalid.
VI_ERROR_INV_SIZE	BFFF007Bh	Invalid size of window specified.
VI_ERROR_WINDOW_MAPPED	BFFF0080h	The specified session currently contains a mapped window.
VI_ERROR_NIMPL_OPER	BFFF0081h	The given operation is not implemented.
VI_ERROR_INV_LENGTH	BFFF0083h	Invalid length specified.
VI_ERROR_INV_MODE	BFFF0091h	The specified mode is invalid.
VI_ERROR_SESN_NLOCKED	BFFF009Ch	The current session did not have any lock on the resource.

VI_ERROR_MEM_NSHARED	BFFF009Dh	The device does not export any memory.
VI_ERROR_LIBRARY_NFOUND	BFFF009Eh	A code library required by VISA could not be located or loaded.
VI_ERROR_NSUP_INTR	BFFF009Fh	The interface cannot generate an interrupt on the requested level or with the requested statusID value.
VI_ERROR_INV_LINE	BFFF00A0h	The value specified by the line parameter is invalid.
VI_ERROR_FILE_ACCESS	BFFF00A1h	An error occurred while trying to open the specified file. Possible reasons include an invalid path or lack of access rights.
VI_ERROR_FILE_IO	BFFF00A2h	An error occurred while performing I/O on the specified file.
VI_ERROR_NSUP_LINE	BFFF00A3h	One of the specified lines (trigSrc or trigDest) is not supported by this VISA implementation, or the combination of lines is not a valid mapping.
VI_ERROR_NSUP_MECH	BFFF00A4h	The specified mechanism is not supported for the given event type.
VI_ERROR_INTF_NUM_NCONFIG	BFFF00A5h	The interface type is valid but the specified interface number is not configured.
VI_ERROR_CONN_LOST	BFFF00A6h	The connection for the given session has been lost.
VI_ERROR_MACHINE_NAVAIL	BFFF00A7h	The remote machine does not exist or is not accepting any connections.
VI_ERROR_NPERMISSION	BFFF00A8h	Access to the resource or remote machine is denied. This is due to lack of sufficient privileges for the current user or machine.

# Completion Codes

This topic lists and describes the completion codes.

Completion Codes	Values	Meaning
VI_SUCCESS	0	Operation completed successfully.
VI_SUCCESS_EVENT_EN	3FFF0002h	Specified event is already enabled for at least one of the specified mechanisms.
VI_SUCCESS_EVENT_DIS	3FFF0003h	Specified event is already disabled for at least one of the specified mechanisms.
VI_SUCCESS_QUEUE_EMPTY	3FFF0004h	Operation completed successfully, but queue was already empty.
VI_SUCCESS_TERM_CHAR	3FFF0005h	The specified termination character was read.
VI_SUCCESS_MAX_CNT	3FFF0006h	The number of bytes read is equal to the input count.
VI_WARN_QUEUE_OVERFLOW	3FFF000Ch	The event returned is valid. One or more events that occurred have not been raised because there was no room available on the queue at the time of their occurrence. This could happen because VI_ATTR_MAX_QUEUE_LENGTH is not set to a large enough value for your application and/or events are coming in faster than you are servicing them.
VI_WARN_CONFIG_NLOADED	3FFF0077h	The specified configuration either does not exist or could not be loaded; using VISA-specified defaults.
VI_SUCCESS_DEV_NPRESENT	3FFF007Dh	Session opened successfully, but the device at the specified address is not responding.
VI_SUCCESS_TRIG_MAPPED	3FFF007Eh	The path from trigSrc to trigDest is already mapped.
VI_SUCCESS_QUEUE_NEMPTY	3FFF0080h	Wait terminated successfully on receipt of an

		event notification. There is still at least one more event occurrence of the requested type(s) available for this session.
VI_WARN_NULL_OBJECT	3FFF0082h	The specified object reference is uninitialized.
VI_WARN_NSUP_ATTR_STATE	3FFF0084h	Although the specified state of the attribute is valid, it is not supported by this resource implementation.
VI_WARN_UNKNOWN_STATUS	3FFF0085h	The status code passed to the operation could not be interpreted.
VI_WARN_NSUP_BUF	3FFF0088h	The specified buffer is not supported.
VI_SUCCESS_NCHAIN	3FFF0098h	Event handled successfully. Do not invoke any other handlers on this session for this event.
VI_SUCCESS_NESTED_SHARED	3FFF0099h	Operation completed successfully, and this session has nested shared locks.
VI_SUCCESS_NESTED_EXCLUSIVE	3FFF009Ah	Operation completed successfully, and this session has nested exclusive locks.
VI_SUCCESS_SYNC	3FFF009Bh	Asynchronous operation request was actually performed synchronously.
VI_WARN_EXT_FUNC_NIMPL	3FFF00A9h	The operation succeeded, but a lower level driver did not implement the extended functionality.

# NI-VISA Driver Wizard Overview

To make your PXI/PCI or USB device visible to NI-VISA applications, the operating system (OS) must know to associate your hardware with the NI-VISA driver. This association is accomplished on Microsoft Windows operating systems using a Setup Information file (`.inf` file).

The NI-VISA Driver Wizard generates one `.inf` file for your PXI/PCI device for use on all supported operating systems. For a USB device, the wizard generates two `.inf` files, one for Windows XP/Server 2003 R2, the other for Windows 7 SP 1 and above. Using the wizard for a USB device is not necessary for use on Linux or Mac OS X. At this time, the list of supported operating systems includes Windows 10/8.1/7 SP1/Vista/XP/Server 2008 R2/Server 2003 R2, LabVIEW RT, Linux, and Mac OS X. The `.inf` file created by the wizard can then be distributed with an instrument driver distribution kit.

## Hardware Bus

Use this page to select which hardware bus is used by the device you want to make visible to NI-VISA applications.



**Note** This wizard is not designed for use with devices that already have an installed device driver.

- **PXI/PCI**—Refers to devices that use either the PCI eXtensions for Instrumentation (PXI) hardware bus or the Personal Computer Interface (PCI) hardware bus.
- **USB**—Refers to devices that use the Universal Serial Bus (USB) hardware bus.

## Basic PXI/PCI Device Information

This dialog contains basic information the operating system needs to locate and to associate your PXI device with the NI-VISA driver software. This information includes essential hardware characteristics that uniquely specify the device, including module

and manufacturer information.

- **Manufacturer Code**—The 16-bit PCI Vendor ID for this device. A list of PCI Vendor ID values is maintained by the PCI Special Interest Group (SIG).
- **Manufacturer Name**—The device manufacturer name.
- **Model Code**—The 16-bit PCI Device ID for this device. The Device ID is uniquely assigned to a PXI/PCI module by the instrument vendor.
- **Model Name**—The PXI/PCI module name.
- **Generates Interrupts**—Check this box if your PXI/PCI device is capable of asserting interrupts and you would like to be able to receive interrupt notification using NI-VISA.
- **Subsystem Manufacturer Code**—The 16-bit PCI Vendor ID used in the subsystem for this device. The PCI Special Interest Group (SIG) maintains a list of PCI Vendor ID values.
- **Subsystem Model Code**—The 16-bit PCI Device ID used in the subsystem for this device. The instrument vendor uniquely assigns the Device ID to a PXI/PCI module.
- **Device Uses Subsystem**—This device has defined values for the PCI subsystem Vendor ID and Device ID.
- **This device uses PXI Express**—PXI Express devices provide to software a way to read the slot number. By checking this box, you can specify the sequence of register accesses necessary to read the slot number from a PXI Express device, or a PXI device that supports this feature.
- **Load Settings from Module Description File**—If a Module Description file (also called a `module.ini` file) is available for this device, you can import the information from that file into the wizard instead of entering the settings yourself. For more information about Module Description files, click the **Help** button.

[More about Module Description Files](#)

## USB Device Selection

This dialog contains a list of all USB devices connected to your system. Clicking a specific device lists all information found in the device, configuration, interface, and endpoint descriptors, along with connection information for the device including the device address and bus speed.

Essential hardware characteristics such as the manufacturer code and manufacturer

name are populated automatically. Other hardware characteristics, such as model name and manufacturer ID, are retrieved if present, but can be overridden if desired.

If your device is not currently connected, you can select the **Other...** option from the device list and enter the information NI-VISA needs manually.



**Note** Using this wizard may not be necessary. NI-VISA may already be able to detect your USB instrument if it conforms to the USB Test & Measurement Class (USBTMC) protocol. If this is the case, DO NOT use this wizard to create an additional `.inf` file.

- **Device List**—Lists all USB devices connected to your system. Select **Other** if you want to manually configure a device that is not currently connected to your system.
- **Connection Information**—Returns connection information for the device you selected in **Device List**, including status, bus speed, and the device address.
- **Descriptor Information**—Returns information obtained from the device, configuration, interface, and endpoint descriptors.
- **USB Manufacturer ID (Vendor ID)**—The 16-bit USB Vendor ID (VID) for this device. A list of USB Vendor ID values is maintained by the USB Implementers Forum (USB-IF).
- **Manufacturer Name**—The device manufacturer name.
- **USB Model Code (Product ID)**—The 16-bit USB Product ID (PID) for this device. The Product ID is uniquely assigned to a USB module by the instrument vendor.
- **Model Name**—The USB module name.

## Basic USB Device Information

Use this page of the [NI-VISA Driver Wizard](#) to view or enter the basic information the operating system needs to locate your USB device and associate it with the NI-VISA driver software. This information includes essential hardware characteristics that uniquely specify the device, including module and manufacturer information.



**Note** Using this wizard may not be necessary. NI-VISA may already be able to detect your USB instrument if it conforms to the USB Test & Measurement Class (USBTMC) protocol. If this is the case, DO NOT use this wizard to create an additional `.inf` file.



- **USB Manufacturer ID (Vendor ID)**—The 16-bit USB Vendor ID (VID) for this device. A list of USB Vendor ID values is maintained by the USB Implementers Forum (USB-IF).
- **Manufacturer Name**—The device manufacturer name.
- **USB Model Code (Product ID)**—The 16-bit USB Product ID (PID) for this device. The Product ID is uniquely assigned to a USB module by the instrument vendor.
- **Model Name**—The USB module name.

## Interrupt Detection Information

Enabling PXI/PCI interrupt handling within an NI-VISA application is a two-step process. First, you must specify how your device [detects a pending interrupt](#). Second, you must specify how to acknowledge a pending interrupt. The NI-VISA Driver Wizard will guide you through the process of enabling NI-VISA to perform these two steps.

- **Add a step before**—Adds a register operation prior to the currently selected step.
- **Add a step after**—Adds a register operation immediately following the currently selected step.
- **Edit a step**—Allows you to modify the properties of the currently selected register operation.
- **Remove a step**—Removes the currently selected step from the sequence of register operations.
- **Select sequence**—Selects a transaction sequence to view or modify.
- **Add sequence**—Adds a new transaction sequence.
- **Remove sequence**—Removes the selected transaction sequence.

## Interrupt Removal Information

In addition to the Interrupt Detection sequence, NI-VISA also needs to know the sequence of register operations required to acknowledge a pending interrupt condition for your device. At interrupt time, if the NI-VISA driver determines that your device is asserting an interrupt (via the sequence of register accesses specified in the Interrupt Detection sequence), VISA will execute this [Interrupt Removal](#) sequence to quiet the pending interrupt.

This sequence of register operations is constructed using the same Read, Write, and

Compare operations discussed in the previous step (Interrupt Detection). Individual register operations are entered in an identical manner.

- **Add a step before**—Adds a register operation prior to the currently selected step.
- **Add a step after**—Adds a register operation immediately following the currently selected step.
- **Edit a step**—Allows you to modify the properties of the currently selected register operation.
- **Remove a step**—Removes the currently selected step from the sequence of register operations.

## NI-VISA PXI Interrupt Information

Use the **Interrupt Information** dialog box to describe an individual register access for the Interrupt Detection sequence or the Interrupt Acknowledge sequence.

- **Type of access**—Register accesses assume the form of reads, writes, or compares.
  - **Read**—Performs a register read (of specified width) from a given offset relative to a given address space.
  - **Write**—Performs a register write of a given value (of specified width) to a given offset relative to a given address space.
  - **Compare**—Performs a Read operation, but additionally applies a user-defined mask to the Read operation result (using a logical AND). The bit mask result is then compared to another user-supplied value. The Compare operation is useful for examining individual bits or combinations of bits within a single register.
  - **NotEqualsCompare**—Performs a Read operation, but additionally applies a user-defined mask to the Read operation result (using a logical AND). The bit mask result is then compared to another user-supplied value. Unlike Compare, however, the command result is considered true only if the comparison is FALSE.
  - **Masked R/W**—Performs a Read operation, then applies a user-defined mask to the Read operation result (using a logical AND). The bit mask result is then written back to the same register in memory.
- **Address space**—The PCI Address space that this operation applies to. Valid address spaces include the PCI Configuration address space (CFG), and any of the Memory or IO address spaces defined in your device's base address registers (BAR0-BAR5).

- **Compare mask**—The mask that will be applied to the result of the register read in a Compare operation. The mask will be logically AND'd with the value read from the register. This field is only valid when a Compare access type is specified.
- **Width of access**—The width (in bits) of the register operation.
- **Offset within space**—The offset within the specified address space to perform this register operation.
- **Value to write or compare**—For a Write operation, this is the value to be written to the register. For a Compare operation, this is the value to compare the masked result with.

## Disarm Interrupt Information

NI-VISA allows you to specify a sequence of register operations to disarm interrupts on your device if a process terminates abnormally. (When a process terminates abnormally, it does not disarm interrupts. This leaves the system vulnerable to receiving an errant interrupt from a device that no longer has an interrupt handler, which could cause a blue screen or system hang.) NI-VISA executes the specified sequence of register operations only if the crashing process is the last process using the device.

## PXI Express Configuration Information

For PXI Express devices, NI-VISA must know the correct register operation sequence to obtain the slot the device is currently plugged into. After you insert a device into a PXI Express chassis and power on the device, the device reads the slot number from pins on the chassis and stores the number in a memory register. To configure the device and chassis automatically, this information must be read from the device memory. Use the NI-VISA Driver Wizard to define how to read this information.

The register operation sequence includes register reads and reads masked with a set value. When using reads masked with a set value, if the mask does not have the least significant bit set, the read value is masked and shifted to the right by the number of least significant bits in the mask that are not set. For example, the mask of 0x78 has binary representation 0b01111000. This reads the register value, masks it with this mask value, and then shifts the result three bits to the right.

## Output Files Information

Use the **Output Files** dialog box to gather the remaining information the NI-VISA Driver Wizard needs to create the Setup Information ( `.inf` ) files. This information includes the instrument driver prefix for your device and the directory where the files should be saved.

- **Instrument Prefix**—Specifies the VXIplug&play-compliant instrument driver prefix for this device. The instrument prefix is used in forming the names of the output files that the wizard generates. The resulting filenames are consistent with instrument driver files created with LabWindows/CVI. If you are not creating a VXIplug&play driver, and you simply need to use VISA to access your device, this field can be left in its default state.
- **Directory in which to save the generated files**—Specifies the directory to save the output files. By default, the wizard saves the files in the instrument driver directory for this device (based on the Instrument Prefix). The default directories are:
  - (Windows) `My Documents\National Instruments\NI-VISA\<Instrument Prefix>`
  - (Linux) `<User Home>/natinst/NI-VISA/<Instrument Prefix>`
  - (Mac OS X) `<User Home>/Documents/National Instruments/NI-VISA/<Instrument Prefix>`
- **Files to be generated**—Displays the files to be generated. The VISA Driver Development Wizard will generate a Setup Information ( `.inf` ) file and, when necessary, a Module Description ( `.ini` ) file.

---

### Related Topics

[Using LabWindows/CVI to Install Your Device .inf Files](#)  
[Creating Your Own Installation Package](#)

## Installation Options

The **Installation Options** dialog offers the following choices for using the Setup Information ( `.inf` ) files after they are generated:

- **Install the generated files on this computer**—Specifies to install the corresponding Setup Information (.inf) file to the local system.
- **FTP the INF file to a LabVIEW RT system**—Specifies to download your device's .inf file to the target system. This option is available only if LabVIEW RT is installed and you are generating Setup Information (.inf) files for PXI/PCI.
- **Take me to the folder containing the generated INI and INF files**—Opens the folder that contains the generated Setup Information (.inf) files.
- **Do nothing further and exit the wizard**—Specifies to exit the wizard without performing any additional tasks.