Tektronix

TekVISA Reference — 071-1104-02

Resource Manager Functions and Operations

`viOpenDefaultRM (ViPSession sesn)`
Open a resource manager session.

`viFindRsrc(ViSession sesn, ViString expr, ViPFindList findlist, ViPUInt32 retCount, ViPRsrc instrdesc)`
Find the first of possibly many instruments.

`viFindNext(ViFindList findlist, ViPRsrc instrdesc)`
Find the next instrument in a list of instruments.

`viOpen(ViSession sesn, ViRsrc rsrcName, ViAccessMode accessmode, ViUInt32 timeout, ViPSession vi)`
Open an instrument session.

`viParseRsrc(ViSession sesn,ViRsrc rsrcName, ViUint16 intfType, ViUInt intfNum)`
Parse a resource string to get the interface information.

Resource Template Operations

`viClose(ViObject vi)`
Close a session (instrument, event, find list, or resource manager).

`viSetAttribute(ViObject vi, ViAttr attribute, ViAttrState attrState)`
Set an attribute; see attributes.

`viGetAttribute(ViObject vi, ViAttr attribute, ViPAttrState attrState)`
Get the current value of an attribute.

`viStatusDesc(ViObject vi, ViStatus status, ViPString desc)`
Convert a status result to a text string.

`viTerminate(ViObject vi, ViUInt16 degree, ViJobId jobId)`
Terminate an asynchronous operation.

`viLock(ViSession vi, ViAccessMode lockType, ViUInt32 timeout, ViKeyId requestedKey, ViPKeyId accessKey)`
Control the access to an instrument.

`viUnlock(ViSession vi)`
Allow others to access an instrument.

`viEventHandler(ViSession vi, ViEventType eventType, ViEvent context, ViAddr userHandle)`
Prototype for callback handler to be called when a particular event occurs.

`viEnableEvent(ViSession vi, ViEventType eventType, ViUInt16 mechanism, ViEventFilter context)`
Allow an event to be reported.

`viDisableEvent(ViSession vi, ViEventType eventType, ViUInt16 mechanism)`
Prevent events from being reported.

`viDiscardEvents(ViSession vi, ViEventType eventType, ViUInt16 mechanism)`
Discard all pending occurrences of an event.

`viWaitOnEvent(ViSession vi, ViEventType inEventType, ViUInt32 timeout, ViPEventType outEventType, ViPEvent outContext)`
Wait for an event to occur.

`viInstallHandler(ViSession vi, ViEventType eventType, ViHndlr handler, ViAddr userHandle)`
Register an event handler.

`viUninstallHandler(ViSession vi, ViEventType eventType, ViHndlr handler, ViAddr userHandle)`
Remove an event handler.

Basic I/O Operations

`viRead(ViSession vi, ViPBuf buf, ViUInt32 count, ViPUInt32 retCount)`
Read from an instrument.

`viReadAsync(ViSession vi, ViPBuf buf, ViUInt32 count, ViPJobId  jobId)`
Read from an instrument but execute while reading.

`viWrite(ViSession vi, ViBuf  buf, ViUInt32 count, ViPUInt32 retCount)`
Write to an instrument.

`viWriteAsync(ViSession vi, ViBuf  buf, ViUInt32 count, ViPJobId  jobId)`
Write to an instrument but execute while writing.

`viAssertTrigger(ViSession vi, ViUInt16 protocol)`
Generate a hardware or software trigger.

`viReadSTB(ViSession vi, ViPUInt16 status)`
Read the status byte.

`viClear(ViSession vi)`
Send a bus-dependent clear command.

`viReadToFile(ViSession vi, ViString filename, ViUInt32 count, ViUInt32 retCount)`
Read data synchronously from a device, and stores the transferred data in a file.

`viWriteFromFile(ViSession vi, ViString filename, ViUInt32 count, ViUInt32 retCount)`
Take data from a file and write it to a device synchronously.

Formatted I/O Operations

`viBufRead(ViSession vi, ViPBuf buf, ViUInt32 count, ViPUInt32 retCount)`
Read data synchronously from a device into the formatted I/O buffer.

`viBufWrite(ViSession vi, ViBuf buf, ViUInt32 count, ViPUInt32 retCount)`
Write data synchronously to a device from the formatted I/O buffer.

`viSetBuf(ViSession vi, ViUInt16 mask, ViUInt32 size)`
Set the size of the formatted I/O and serial I/O buffers.

`viFlush(ViSession vi, ViUInt16 mask)`
Empty a formatted I/O or serial I/O buffer.

`viPrintf(ViSession vi, ViString writeFmt,…)`
Create a formatted string and send it to an instrument.

`viSPrintf(ViSession vi, ViPBuf buf, ViString writeFmt,…)`
Create a formatted string and send it to an instrument using a user-supplied buffer.

`viVPrintf(ViSession vi, ViString writeFmt, ViVAList params)`
Create a formatted string and send it to an instrument using a pointer.

`viVSPrintf(ViSession vi, ViPBuf buf, ViString writeFmt,…)`
Create a formatted string and send it to an instrument using a pointer and a user-supplied buffer.

`viScanf(ViSession vi, ViString readFmt, …)`
Read and extract data from an instrument. Perform formatted input.

`viSScanf(ViSession vi, ViPBuf buf, ViString readFmt,…)`
Read and extract data from an instrument. Perform formatted input using a user-supplied buffer.

`viVScanf(ViSession vi, ViString readFmt, ViVAList params)`
Read and extract data from an instrument. Perform formatted input using a pointer.

`viVSScanf(ViSession vi, ViPBuf buf, ViString readFmt,…)`
Read and extract data from an instrument. Perform formatted input using a user-supplied buffer.

`viQueryf(ViSession vi, ViString writeFmt, ViString readFmt,…)`
Write formatted data to and read formatted data from an instrument.

`viVQueryf(ViSession vi, ViString writeFmt, ViString readFmt, ViVAList params);`
Write formatted data to and read formatted data from an instrument using a pointer.

Attribute	Type	R/W
VI_ATTR_ASRL_AVAIL_NUM	ViUInt32	RO
VI_ATTR_ASRL_BAUD	ViUInt32	RW
VI_ATTR_ASRL_CTS_STATE	ViInt16	RO
VI_ATTR_ASRL_DATA_BITS	ViUInt16	RW
VI_ATTR_ASRL_DCD_	ViInt16	RO
VI_ATTR_ASRL_DSR_STATE	ViInt16	RO
VI_ATTR_ASRL_DTR_STATE	ViInt16	RW
VI_ATTR_ASRL_END_IN	ViUInt16	RW
VI_ATTR_ASRL_END_OUT	ViUInt16	RW
VI_ATTR_ASRL_FLOW_CNTRL	ViUInt16	RW
VI_ATTR_ASRL_PARITY	ViUInt16	RW
VI_ATTR_ASRL_REPLACE_CHAR	ViUInt8	RW
VI_ATTR_ASRL_RI_STATE	ViInt16	RO
VI_ATTR_ASRL_RTS_STATE	ViInt16	RW
VI_ATTR_ASRL_STOP_BITS	ViUInt16	RW
VI_ATTR_ASRL_XOFF_CHAR	ViUInt8	RW
VI_ATTR_ASRL_XON_CHAR	ViUInt8	RW
VI_ATTR_BUFFER	ViBuf	RO
VI_ATTR_EVENT_TYPE	ViEventType	RO
VI_ATTR_FILE_APPEND_EN	Boolean	RW
VI_ATTR_GPIB_PRIMARY_ADDR	ViUInt16	RO
VI_ATTR_GPIB_READDR_EN	ViBoolean	RW
VI_ATTR_GPIB_SECONDARY_ADDR	ViUInt16	RO
VI_ATTR_GPIB_UNADDR_EN	ViBoolean	RW
VI_ATTR_INTF_INST_NAME	ViString	RO
VI_ATTR_INTF_NUM	ViUInt16	RO
VI_ATTR_INTF_TYPE	ViUInt16	RO
VI_ATTR_IO_PROT	ViUInt16	RW
VI_ATTR_JOB_ID	ViJobID	RO
VI_ATTR_MAX_QUEUE_LENGTH	ViUInt32	RW
VI_ATTR_OPER_NAME	ViString	RO
VI_ATTR_RD_BUF_OPER_MODE	ViUInt16	RW
VI_ATTR_RET_COUNT	ViUInt32	RO
VI_ATTR_RM_SESSION	ViSession	RO
VI_ATTR_RSRC_IMPL_VERSION	ViVersion	RO
VI_ATTR_RSRC_LOCK_STATE	ViAccessMode	RO
VI_ATTR_RSRC_MANF_ID	ViUInt16	RO
VI_ATTR_RSRC_MANF_NAME	ViString	RO
VI_ATTR_RSRC_NAME	ViRsrc	RO
VI_ATTR_RSRC_SPEC_VERSION	ViVersion	RO
VI_ATTR_SEND_END_EN	ViBoolean	RW
VI_ATTR_STATUS	ViStatus	RO
VI_ATTR_SUPPRESS_END_EN	ViBoolean	RW
VI_ATTR_TCPIP_ADDR	String	RO
VI_ATTR_TCPIP_HOSTNAME	String	RO
VI_ATTR_TERMCHAR	ViUInt8	RW
VI_ATTR_TERMCHAR_EN	ViBoolean	RW
VI_ATTR_TMO_VALUE	ViUInt32	RW
VI_ATTR_TRIG_ID	ViUInt16	RW
VI_ATTR_USER_DATA	ViAddr	RW
VI_ATTR_WR_BUF_OPER_MODE	ViUInt16	RW

Event Types

VI_EVENT_EXCEPTION
VI_EVENT_IO_COMPLETION
VI_EVENT_SERVICE_REQ

Completion and Error Codes

- VI_SUCCESS — The operation completed successfully.
- > VI_SUCCESS — The operation succeeded conditionally. This return condition may need to be handled. See TekVISA manual for more information.
- < VI_SUCCESS — The operation failed.

A Read/Write Example

```
#include <visa.h>
#include <stdio.h>

int main(int argc, char* argv[]) {
    ViSession rm, vi;
    ViUInt32 retCnt;
    ViChar buffer[256];

    viOpenDefaultRM(&rm);

    viOpen(rm, “GPIB0::1::INSTR”, VI_NULL,
           VI_NULL, &vi);

    viWrite(vi, “*idn?”, 5, &retCnt);
    viRead(vi, buffer, 256, &retCnt);

    printf(“device: %s\n”, buffer);

    viClose(rm);
}
```

An Attribute Example

```
#include <visa.h>
#include <stdio.h>

int main(int argc, char* argv[]) {
    ViSession  rm, vi;
    ViChar     buffer[256];

    viOpenDefaultRM(&rm);

    viOpen(rm, “GPIB0::1::INSTR”, VI_NULL,
           VI_NULL, &vi);

    //Get VISA Manufacturer Name
    viGetAttribute(vi, VI_RSRC_MANF_NAME,
                  (ViPAttrState) buffer);

    // Set Timeout to 5 seconds
    viSetAttribute(vi, VI_ATTR_TMO_VALUE,
                   5000);

    printf(“Manufacturer: %s\n”, buffer);

    viClose(rm);
}
```

An Exclusive Lock Example

```
#include <visa.h>
#include <stdio.h>

int main(int argc, char* argv[]) {
    ViSession rm, vi;
    ViUInt32 retCnt;
    ViChar buffer[256];

    viOpenDefaultRM(&rm);

    viOpen(rm, “GPIB0::1::INSTR”, VI_NULL,
           VI_NULL, &vi);

    // Locking the read/write ensures a
    // second application talking to the
    // same resource works as expected.
    viLock(vi, VI_EXCLUSIVE_LOCK,
           VI_TMO_INFINITE, VI_NULL,
           VI_NULL);

    viWrite(vi, “*idn?”, 5, &retCnt);
    viRead(vi, buffer, 256, &retCnt);

    viUnlock(vi);

    printf(“device: %s\n”, buffer);

    viClose(rm);
}
```

A Formatted I/O Example (See † Note)

```
#include <visa.h>
#include <stdio.h>

int main(int argc, char* argv[]) {
    ViSession rm, vi;
    ViChar buffer[256];

    viOpenDefaultRM(&rm);

    viOpen(rm, “GPIB0::1::INSTR”, VI_NULL,
           VI_NULL, &vi);

    viPrintf(vi, “header off”);
    viFlush(vi, VI_WRITE_BUF);

    // No locking is required when
    // using viQuery
    viQueryf(vi, “*idn?”, “%s”, buffer);

    printf(“device: %s\n”, buffer);

    viClose(rm);
}
```

TekVISA Version 3.0
09/27/06