Application Note AN1001

How to use AnaPico API libraries

Purpose

This application note describes how to use the communication API's provided by our C and COM libraries.

The communication libraries support LAN, USB and GPIB links and work for all AnaPico devices.

Contents

1. Introduction

AnaPico libraries provide communication API's for all AnaPico devices. The libraries support LAN, USB and GPIB connectivity.

USB support may require a manual USBTMC driver installation. This driver is part of the VISA shared components provided by the IVI foundation.

GPIB support requires a third party VISA installation, such as NI VISA or Agilent IO Libraries Suite.

This application note describes how to communicate with an instrument. Our devices provide different command sets reflecting their various capabilities. Please refer to the device specific programmer's manual for information about the command set.

2. AnaPico C library

The C library can be integrated in virtually any language. The C library is provided for both 32 and 64 bit Microsoft Windows systems.
The functions follow the CDECL calling convention.

2.1. Files

ap_comm_lib.h	Header file describing the API. Add this file to your compiler's includes.
ap_comm_lib.lib	Import library file. 32 bit version. Add this file to your linker's dependencies.
ap_comm_lib_64.lib	Import library file. 64 bit version. Add this file to your linker's dependencies.
ap_comm_lib.dll	Dynamically-linked library file. 32 bit version. This file needs to be in the same folder as your executable.
ap_comm_lib_64.dll	Dynamically-linked library file. 64 bit version. This file needs to be in the same folder as your executable.
VisaSharedComponents32_1.6.exe	VISA shared components including the USB (USBTMC) driver. 32 bit version.
VisaSharedComponents64_1.6.exe	VISA shared components including the USB (USBTMC) driver. 64 bit version.

2.2. Signal generator control example (C)

This C example shows how to

- find a device connected to your PC,
- open and close a connection,
- send commands and receive responses,
- send and receive block data (a sweep list).

To compile and run this example:

- Add the ap_comm_lib.h header file to your compiler's includes.
- Add the ap_comm_lib.lib or ap_comm_lib_64.lib (depending on your system) file to your linker's dependencies.
- The ap_comm_lib.dll or ap_comm_lib_64.dll (depending on your system) must be located in the executable's directory.

```
#include <string.h>
#include <stdio.h>
#include "ap_comm_lib.h"
```

```
#define DEVICES_BUFFER_SIZE 1000
#define BUFFER_SIZE 1000
#define MAX_DEVICES 20

int main(void)
{
	char* str;
	char* device[MAX_DEVICES];
	char devices[DEVICES_BUFFER_SIZE];
	int nDevices;
	int retCount;
	int selection;
	char* address;
	int linkId;
	char buffer[BUFFER_SIZE];

	ap_comm_loadVisa();

	ap_comm_find(devices,
		     1000,
		     DEVICES_BUFFER_SIZE,
		     NULL,
		     AP_COMM_IFACE_LAN
		     |AP_COMM_IFACE_USB
		     |AP_COMM_IFACE_VISA_GPIB);

	// Print devices list
	printf("Devices list:\n");
	str = devices;
	for (nDevices=0; nDevices<MAX_DEVICES; nDevices++) {
		device[nDevices] = str;
		str = strchr( str, ';' );
		if (str!=NULL) {
			str[0] = '\0';
			str++;
		} else {
			break;
		}
		printf("  %d %s\n", nDevices, device[nDevices]);
	}
	// Return if no devices foun
	if (nDevices==0) {
		printf("No devices found\n");
		getchar();
		ap_comm_freeVisa();
		return 0;
	}

	// Let user select a device
	printf("Select a device: ");
```

```
	scanf_s( "%d", &selection );
	getchar();
	// Skip model within device string
	if (selection<nDevices) {
		str = strchr( device[selection], ':' );
	}
	// Skip serial number within device string
	if (str!=NULL) {
		str = strchr( str+1, ':' );
	}
	// Get address
	if (str!=NULL) {
		address = str+1;
	}

	// Open link
	ap_comm_open(address, 2000, &linkId);

	// Query "*IDN?"
	ap_comm_puts(linkId, "*IDN?\n", 0, NULL);
	ap_comm_gets(linkId, buffer, BUFFER_SIZE, NULL);
	printf("*IDN? response:\n  %s\n", buffer);

	// Send sweep list
	sprintf_s(buffer,
			BUFFER_SIZE,
			"11e6;0.0;1.0;1.0\n"
			"22e6;0.0;1.0;1.0\n"
			"33e6;0.0;1.0;1.0\n"
			"44e6;0.0;1.0;1.0\n"
			"55e6;0.0;1.0;1.0\n");
	ap_comm_writeBlock(linkId,
					"MEM:FILE:LIST:DATA\n",
					buffer,
					strlen(buffer));

	// Query sent list's number of points
	ap_comm_puts(linkId, "LIST:CURR:POIN?\n", 0, NULL);
	ap_comm_gets(linkId, buffer, BUFFER_SIZE, NULL);
	printf("Device received %s points list\n\n", buffer);

	// Receive processed sweep list
	ap_comm_readBlock(linkId,
				"MEM:FILE:LIST:DATA?\n",
				buffer,
				BUFFER_SIZE,
				&retCount);
	buffer[retCount] = '\0';
	printf("Processed list data:\n%s", buffer);

	// Close link
```

```
        ap_comm_close(linkId);
        ap_comm_freeVisa();
        getchar();
        return 0;
}
```

2.3. C library reference

Common success and error return codes

VI_SUCCESS	0x0	Operation completed successfully.
VI_SUCCESS_MAX_CNT	0x3FFF0006	The number of bytes read is equal to the input count, more data available.
VI_ERROR_SYSTEM_ERROR	0xBFFF0000	Unknown system error
VI_ERROR_TMO	0xBFFF0015	Timeout expired before operation completed.
VI_ERROR_CONN_LOST	0xBFFF00A6	The connection for the given session has been lost.
VI_ERROR_FILE_ACCESS	0xBFFF00A1	Accessing the file system failed.

2.3.1. ap_comm_loadVisa

int ap_comm_loadVisa(void)

Loads the external VISA libraries and enables VISA support. VISA support is required to communicate with GPIB instruments.

Returns:

VI_SUCCESS in case of success, VI_ERROR_SYSTEM_ERROR if no installed VISA library has been found.

2.3.2. ap_comm_freeVisa

void ap_comm_freeVisa(void)

Frees the external VISA libraries and disables VISA support.

2.3.3. ap_comm_find

int ap_comm_find(char* devices,
int tmoMs,
int count,
int* retCount,
int iface)

Finds AnaPico devices.

Parameters:

out	devices	Buffer for the found devices string. String format: Model1:SerNo1:Addr1;Model2:SerNo2:Addr2;...
in	tmoMs	Find timeout in milliseconds, e.g. 500. Decrease it to speed up the find process, increase it if find fails sometimes.
in	count	Size of the devices buffer, in bytes.
out	retCount	Returns the number of bytes written to the devices buffer.
in	iface	Interface selection, bitwise OR of one or more of the following macros: AP_COMM_IFACE_LAN (LAN) AP_COMM_IFACE_USB (USB)

AP_COMM_IFACE_VISA_GPIB (GPIB, requires a VISA installation and VISA enabled by calling ap_comm_loadVisa())

Returns:
VI_SUCCESS in case of success, VI_ERROR code otherwise.

2.3.4. ap_comm_open

```
int ap_comm_open(   char* address,
                    int tmoMs,
                    int* retLinkId    )
```

Opens a connection to an AnaPico USB device. Pass an address returned by ap_comm_find() or create an address string manually.

LAN address format: **"[TCP-]<IPv4 address>"** where **<IPv4 address>** is the device IP address, e.g. "TCP-192.168.1.2" or "192.168.1.2".

USB address format: **"USB-<SN>"** where **<SN>** is the serial number, e.g. "USB-121-213300010-0093".

VISA GPIB address format: **"GPIB<IF no>::<GPIB address>::INSTR"** where **<IF no>** is the VISA GPIB interface number and **<GPIB address>** is the GPIB device address, e.g. "GPIB0::1::INSTR"

Parameters:

in	address	Device address
in	tmoMs	Initial connection timeout in milliseconds, e.g. 2000.
out	retLinkId	Link ID >=1 in case of success, -1 otherwise.

Returns:
VI_SUCCESS in case of success, VI_ERROR code otherwise.

2.3.5. ap_comm_close

```
int ap_comm_close( int linkId )
```

Closes a connection to an AnaPico USB device.

Parameters:

in	linkId	Link ID returned by ap_comm_open().

Returns:
VI_SUCCESS in case of success, VI_ERROR code otherwise.

2.3.6. ap_comm_puts

```
int ap_comm_puts(   int linkId,
                    char* cmd,
                    int count,
                    int* retCount    )
```

Sends a command or query to an AnaPico device.

Parameters:

in	linkId	Link ID returned by ap_comm_open().
in	cmd	Command string. Can be terminated by '\0' or '\n'.
in	count	Size of the command string, in bytes. Zero can be passed if the command string is terminated by '\0' or '\n'.
out	retCount	Returns the number of bytes actually sent to the device. NULL

can be passed.

Returns:
VI_SUCCESS in case of success, VI_ERROR code otherwise.

int ap_comm_gets(int linkId,
char* ans,
int count,
int* retCount)

Receives a query response from an AnaPico device.

Parameters:

in	linkId	Link ID returned by ap_comm_open().
out	ans	Response buffer. Responses are terminated by '\n' and '\0'.
in	count	Size of the response buffer, in bytes.
out	retCount	Returns the number of bytes actually read from to the device. NULL can be passed.

Returns:
VI_SUCCESS in case of success, VI_ERROR code otherwise.

2.3.7. ap_comm_write

int ap_comm_write(int linkId,
char* buffer,
int count,
int* retCount)

Sends binary data to an AnaPico device.
Unlike ap_comm_puts(), this function does not care about command termination.

Parameters:

in	linkId	Link ID returned by ap_comm_open().
in	buffer	Data.
in	count	Data size in bytes.
out	retCount	Returns the number of bytes actually sent to the device. NULL can be passed.

Returns:
VI_SUCCESS in case of success, VI_ERROR code otherwise.

2.3.8. ap_comm_read

int ap_comm_read(int linkId,
char* buffer,
int count,
int* retCount)

Reads binary response data from an AnaPico USB device.
Unlike ap_comm_gets(), this function does not care about response termination.
It is not guaranteed that this function returns a complete response. Thus it's in the callers responsibility to repeat the read calls until all data has been read.

Parameters:

in	linkId	Link ID returned by ap_comm_open().

out	buffer	Response data buffer.
in	count	Data size in bytes.
out	retCount	Returns the number of bytes actually from the device.

Returns:

VI_SUCCESS in case of success, VI_ERROR code otherwise.

2.3.9. ap_comm_writeBlock

```
int ap_comm_writeBlock(   int linkId,
                          char* cmd,
                          void* buffer,
                          int size        )
```

Sends a binary data block to an AnaPico device. The data is retrieved from a buffer. Uses the IEEE488.2 definite length arbitrary block data format.

Parameters:

in	linkId	Link ID returned by ap_comm_open().
in	cmd	Command preceding the binary data block.
in	buffer	Data to be sent.
in	size	Number of data bytes to be sent.

Returns:

VI_SUCCESS in case of success, VI_ERROR code otherwise.

2.3.10. ap_comm_writeBlockFile

```
int ap_comm_writeBlockFile(   int linkId,
                              char* cmd,
                              char* filename  )
```

Sends a binary data block to an AnaPico device. The data is retrieved from a file. Uses the IEEE488.2 definite length arbitrary block data format.

Parameters:

in	linkId	Link ID returned by ap_comm_open().
in	cmd	Command preceding the binary data block.
in	filename	String containing path and name of file containing data to be sent.

Returns:

VI_SUCCESS in case of success, VI_ERROR code otherwise.

2.3.11. ap_comm_readBlockSize

```
int ap_comm_readBlockSize(   int linkId,
                             char* cmd,
                             int* retSize      )
```

Returns the size of a binary data block from an AnaPico device. Call ap_comm_readBlockData() to get the block data itself. Supports both IEEE488.2 definite length arbitrary block and AnaPicos proprietary block data formats.

Parameters:

in	linkId	Link ID returned by ap_comm_open().
in	cmd	Command returning the binary data block.

out	retSize	Size of the binary data block in bytes.

Returns:

VI_SUCCESS in case of success, VI_ERROR code otherwise.

2.3.12. ap_comm_readBlockData

```
int ap_comm_readBlockData(  int linkId,
                            void* buffer,
                            int size,
                            int* retSize      )
```

Reads block data from an AnaPico device after querying the block size by calling ap_comm_readBlockSize(). The data will be written to a buffer. Supports both IEEE488.2 definite length arbitrary block and AnaPicos proprietary block data formats.

Parameters:

in	linkId	Link ID returned by ap_comm_open().
in	buffer	Buffer holding received data.
in	size	Size of the buffer in bytes.
out	retSize	Number of bytes received.

Returns:

VI_SUCCESS or VI_SUCCESS_MAX_CNT in case of success, VI_ERROR code otherwise. VI_SUCCESS_MAX_CNT indicates that there is more data available. Call this function repeatedly until it returns VI_SUCCESS.

2.3.13. ap_comm_readBlock

```
int ap_comm_readBlock(      int linkId,
                            char* cmd,
                            void* buf,
                            int count,
                            int* retCount    )
```

Reads a binary data block from an AnaPico device. The data will be written to a buffer. Supports both IEEE488.2 definite length arbitrary block and AnaPicos proprietary block data formats. Calling ap_comm_readBlock() is equal to calling ap_comm_readBlockSize() and ap_comm_readBlockData().

Parameters:

in	linkId	Link ID returned by ap_comm_open().
in	cmd	Command returning the binary data block.
in	buffer	Buffer holding received data.
in	size	Size of the buffer in bytes.
out	retSize	Number of bytes received.

Returns:

VI_SUCCESS or VI_SUCCESS_MAX_CNT in case of success, VI_ERROR code otherwise. VI_SUCCESS_MAX_CNT indicates that there is more data available. Call this function repeatedly until it returns VI_SUCCESS.

2.3.14. ap_comm_readBlockFile

```
int ap_comm_readBlockFile(  int linkId,
                            char* cmd,
```

char* filename)

Reads a binary data block from an AnaPico device. The data will be written to a file. Supports both IEEE488.2 definite length arbitrary block and AnaPicos proprietary block data formats.

Parameters:

in	linkId	Link ID returned by ap_comm_open().
in	cmd	Command returning the binary data block.
in	filename	Path and name of file to be stored. An existing file will be overwritten.

Returns:

VI_SUCCESS in case of success, VI_ERROR code otherwise.

2.3.15. ap_comm_clear

int ap_comm_clear(int linkId)

Clears all link buffers. Clears data pending to be sent or received.

Parameters:

in	linkId	Link ID returned by ap_comm_open().

Returns:

VI_SUCCESS in case of success, VI_ERROR code otherwise.

2.3.16. ap_comm_setTmoMs

int ap_comm_setTmoMs(int linkId,
int tmoMs)

Sets the connection timeout in milliseconds.

Parameters:

in	linkId	Link ID returned by ap_comm_open().
in	tmoMs	Timeout in milliseconds. Default is 2000.

Returns:

VI_SUCCESS in case of success, VI_ERROR code otherwise.

3. AnaPico COM library

The AnaPico COM (component object model) library provides a communication API for all AnaPico devices.

COM objects can easily be integrated in many object oriented languages such as C++, C#, VB.net, VBA and many more.

3.1. Files

register_AP_COMM_COM.bat	This batch file registers the COM server.
ap_comm_com.dll	COM server (library file). This file needs to be registered by running the register_AP_COMM_COM.bat batch file or by calling regsvr32.exe.
ap_comm_lib.dll	Dynamically-linked library file. Required by the COM server.
VisaSharedComponents32_1.6.exe	VISA shared components including the USB (USBTMC) driver. 32 bit version.
VisaSharedComponents64_1.6.exe	VISA shared components including the USB (USBTMC) driver. 64

bit version.

3.2. Signal generator control example (VBA)

This VBA example shows how to

- find a device connected to your PC,
- open and close a connection,
- send commands and receive responses,
- send and receive block data (a sweep list).

To run this example:

- Register the AP_COMM_COM library by running the register_AP_COMM_COM.bat batch file.
- Within your project, include the "AP_COMM_COM 1.0 Type Library". In Microsoft Visual Basic: Tools -> References.

```
Sub example()

' Load library (loading of VISA (GPIB) support is optional)
Dim COMM As AP_COMM_COMLib.COMM
Set COMM = New AP_COMM_COMLib.COMM
COMM.loadVisa

' Find devices
Dim devices As String
Call COMM.Find(1000, LAN Or USB Or VISA_GPIB, devices)
' Separate found devices
Dim devicesArray() As String
devicesArray = Split(devices, ";")
' Separate fields of first device
Dim firstDeviceFields() As String
firstDeviceFields = Split(devicesArray(0), ":", 3)
' Get address string of first device
Dim firstDeviceAddress As String
firstDeviceAddress = firstDeviceFields(2)
MsgBox ("Address of first device found: " + firstDeviceAddress)

' Open link
Dim link As AP_COMM_COMLib.instr
Set link = COMM.openinstr(firstDeviceAddress, 2000)

' Query "*IDN?"
link.puts ("*IDN?")
Dim ans As String
ans = link.gets
MsgBox ("*IDN? response: <" + ans + ">")

' Send sweep list
Dim list As String
list = "10e6;0.0;1.0;1.0" + vbLf _
     + "20e6;0.0;1.0;1.0" + vbLf _
     + "30e6;0.0;1.0;1.0" + vbLf _
     + "40e6;0.0;1.0;1.0" + vbLf _
```

```
     + "50e6;0.0;1.0;1.0" + vbLf
Dim listData() As Byte
listData = StrConv(list, vbFromUnicode)
Call link.writeBlock("MEM:FILE:LIST:DATA", listData)

' Query sent list's number of points
link.puts ("LIST:CURR:POIN?")
ans = link.gets
MsgBox ("Device received " + ans + " points list")

' Receive processed sweep list
listData = link.readBlock("MEM:FILE:LIST:DATA?", BlockType_UI1)
list = StrConv(listData, vbUnicode)
MsgBox ("Processed list data:" + vbLf + list)

' Close link
link.Close
Set link = Nothing
COMM.freeVisa
Set COMM = Nothing
End Sub
```

3.3. COM library reference

3.3.1. Common success and error return codes

S_VI_SUCCESS	0x40000	Operation completed successfully.
S_VI_SUCCESS_MAX_CNT	0x40006	The number of bytes read is equal to the input count, more data available.
E_VI_ERROR_SYSTEM_ERROR	0x80040000	Unknown system error
E_VI_ERROR_TMO	0x80040015	Timeout expired before operation completed.
E_VI_ERROR_CONN_LOST	0x800400A6	The connection for the given session has been lost.
E_VI_ERROR_FILE_ACCESS	0x800400A1	Accessing the file system failed.
E_UNSUPPORTED_TYPE	0x8007065E	Requested data type is not supported.
E_DATATYPE_MISMATCH	0x8007065D	Data block size doesn't match the requested data type size. Data block size (bytes) must be a multiple of the data type size.
E_OUTOFMEMORY	0x8007000E	Internal memory allocation failed.

3.3.2. IFACE enumeration

The IFACE enumeration selects between the available interfaces. Combine the values bitwise to select multiple interfaces.

LAN	Selects the LAN (network) interface.
USB	Selects the USB interface. The USB drivers must be installed properly.
VISA_GPIB	Selects the GPIB interface. A VISA installation must be available and VISA support must be loaded before by calling COMM::loadVisa.

3.3.3. BLOCKTYPE enumeration

The BLOCKTYPE enumeration specifies the type of data read from an AnaPico device. The programmer's manual describes the type of block data returned upon a certain query. Usually, ASCII list data (power correction, sweep lists) is BlockType_UI1 (VB: Byte, C++: unsigned char) and real (measurement) data (phase noise) is BlockType_R4 (VB: Single, C++: 32 bit float).

BlockType_UI1	8 bit unsigned integer format. Used for ASCII list data.
BlockType_I2	16 bit signed integer format.
BlockType_I4	32 bit signed integer format.
BlockType_R4	32 bit IEEE754 floating point format. Used for real (measurement) data.
BlockType_R8	64 bit IEEE754 floating point format.

3.3.4. COMM interface

This is the base interface for finding and opening AnaPico devices.

3.3.5. COMM::loadVisa

VB prototype:

loadVisa()

C++ prototype:

HRESULT loadVisa()

Loads the external VISA libraries and enables VISA support. VISA support is required to communicate with GPIB instruments.

Returns:

VI_SUCCESS in case of success, VI_ERROR_SYSTEM_ERROR if no installed VISA library has been found.

3.3.6. COMM::freeVisa

VB prototype:

freeVisa()

C++ prototype:

HRESULT freeVisa()

Frees the external VISA libraries and disables VISA support.

Returns:

S_OK.

3.3.7. COMM::find

VB prototype:

```
devices as String = find(       tmoMs as LONG,
                                iface as IFACE    )
```

C++ prototype:

```
HRESULT find(       long tmoMs,
                    IFACE iface,
                    BSTR* devices   )
```

Finds AnaPico devices.

Parameters:

in	tmoMs	Find timeout in milliseconds, e.g. 500. Decrease it to speed up the find process, increase it if find fails sometimes.
in	iface	Interface selection, bitwise OR of one or more of the following macros: AP_COMM_IFACE_LAN (LAN) AP_COMM_IFACE_USB (USB) AP_COMM_IFACE_VISA_GPIB (GPIB, requires a VISA installation and VISA enabled by calling ap_comm_loadVisa())

Returned data:

devices	Buffer holding the found devices string. String format: Model1:SerNo1:Addr1;Model2:SerNo2:Addr2;...

Returns:

VI_SUCCESS in case of success, VI_ERROR code otherwise.

3.3.8. COMM::openinstr

VB prototype:

```
inst as INSTR = openinstr(      address as String,
                                tmoMs as LONG       )
```

C++ prototype:

```
HRESULT openinstr(      BSTR address,
                        long tmoMs,
                        INSTR** inst    )
```

Opens a connection to an AnaPico USB device. Pass an address returned by COMM::find() or create an address string manually.

LAN address format: **"[TCP-]<IPv4 address>"** where <IPv4 address> is the device IP address, e.g. "TCP-192.168.1.2" or "192.168.1.2".

USB address format: **"USB-<SN>"** where <SN> is the serial number, e.g. "USB-121-213300010-0093".

VISA GPIB address format: **"GPIB<IF no>::<GPIB address>::INSTR"** where <IF no is the VISA GPIB interface number and <GPIB address> is the GPIB device address, e.g. "GPIB0::1::INSTR"

Parameters:

in	address	Device address.
in	tmoMs	Initial connection timeout in milliseconds, e.g. 2000.

Returned data:

inst	The new link to an AnaPico device.

Returns:

VI_SUCCESS in case of success, VI_ERROR code otherwise.

3.3.9. INSTR interface

This is the interface that provides a link to an AnaPico device.

3.3.10. INSTR::puts

VB prototype:

```
puts(   str As String   )
```

C++ prototype:

HRESULT puts(BSTR str)

Sends a command or query to an AnaPico device.

Parameters:

in	str	Command string.

Returns:

VI_SUCCESS in case of success, VI_ERROR code otherwise.

3.3.11. INSTR::gets

VB prototype:

str As String = gets()

C++ prototype:

HRESULT gets(BSTR* str)

Receives a query response from an AnaPico device.

Returned data:

str	Response string.

Returns:

VI_SUCCESS in case of success, VI_ERROR code otherwise.

3.3.12. INSTR::write

VB prototype:

```
retCount As Long = write(    buffer() As Byte,
                             count As Long   )
```

C++ prototype:

```
HRESULT write(    SAFEARRAY(BYTE)* buffer,
                  long count,
                  long* retCount              );
```

Sends binary data to an AnaPico device.

Parameters:

in	buffer	Data.
in	count	Data size in bytes.

Returned data:

retCount	Number of bytes actually sent to the device.

Returns:

VI_SUCCESS in case of success, VI_ERROR code otherwise.

3.3.13. INSTR::read

VB prototype:

buffer() As Byte = read(count As Long)

C++ prototype:

```
HRESULT read(     long count
                  SAFEARRAY(BYTE)* buffer     )
```

Reads binary response data from an AnaPico device.
Unlike ap_comm_gets(), this function does not care about response termination.
It is not guaranteed that this function returns a complete response. Thus it's in the callers responsibility to repeat the read calls until all data has been read.

Parameters:

in	count	Maximum number of bytes to be read.

Returned data:

buffer	Data read from device.

Returns:
VI_SUCCESS in case of success, VI_ERROR code otherwise.

3.3.14. INSTR::writeBlock

VB prototype:

```
writeBlock(     cmd As String,
                data as variant  )
```

C++ prototype:

```
HRESULT read(           BSTR cmd,
                        VARIANT data   )
```

Sends a binary data block to an AnaPico device. The data is retrieved from a buffer. Uses the IEEE488.2 definite length arbitrary block data format.

Parameters:

in	cmd	Command preceding the binary data block.
in	data	Data to be sent.

Returns:
VI_SUCCESS in case of success, VI_ERROR code otherwise.

3.3.15. INSTR::readBlock

VB prototype:

```
data As Variant = readBlock(    cmd As String,
                                type As BLOCKTYPE       )
```

C++ prototype:

```
HRESULT readBlock(      BSTR cmd,
                        BLOCKTYPE type,
                        VARIANT* data )
```

Reads a binary data block from an AnaPico device. The data will be written to a buffer. Supports both IEEE488.2 definite length arbitrary block and AnaPicos proprietary block data formats.

Parameters:

in	cmd	Command returning the binary data block.
in	type	Data type of returned data. Refer to the BLOCKTYPE enumeration description.

Returned data:

data	Data read from device.

Returns:
VI_SUCCESS in case of success, VI_ERROR code otherwise.

3.3.16. INSTR::writeBlockFile

VB prototype:

```
writeBlockFile(     cmd As String,
                    filename As String      )
```

C++ prototype:

```
HRESULT writeBlockFile(     BSTR cmd,
                            BSTR filename   )
```

Sends a binary data block to an AnaPico device. The data is retrieved from a file. Uses the IEEE488.2 definite length arbitrary block data format.

Parameters:

in	cmd	Command preceding the binary data block.
in	filename	String containing path and name of file containing data to be sent.

Returns:

VI_SUCCESS in case of success, VI_ERROR code otherwise.

3.3.17. INSTR::readBlockFile

VB prototype:

```
readBlockFile(      cmd As String,
                    filename As String      )
```

C++ prototype:

```
HRESULT readBlockFile(      BSTR cmd,
                            BSTR filename   )
```

Reads a binary data block from an AnaPico device. The data will be written to a file. Supports both IEEE488.2 definite length arbitrary block and AnaPicos proprietary block data formats.

Parameters:

in	cmd	Command preceding the binary data block.
in	filename	String containing path and name of file containing data to be sent.

Returns:

VI_SUCCESS in case of success, VI_ERROR code otherwise.

3.3.18. INSTR::clear

VB prototype:

```
clear()
```

C++ prototype:

```
HRESULT clear()
```

Clears all link buffers. Clears data pending to be sent or received.

Returns:

VI_SUCCESS in case of success, VI_ERROR code otherwise.

3.3.19. INSTR::setTmoMs

VB prototype:

setTmoMs(tmoMs As Long)

C++ prototype:

setTmoMs(long tmoMs)

3.3.20. INSTR::close

VB prototype:

close()

C++ prototype:

close()

Closes a connection to an AnaPico device. The INSTR instance can be deleted after calling close().

Returns:

VI_SUCCESS in case of success, VI_ERROR code otherwise.