Programming Samples (Mixed Devices) User's Manual

For the

New Focus Picomotor Application

Version 1.0.0

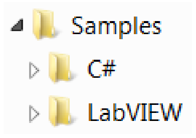
Prepared by:
New Focus
3635 Peterson Way
Santa Clara, CA 95054

Table of Contents

1 Introduction

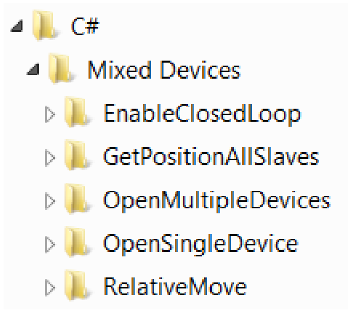
The samples for the Picomotor controller are divided into two folders: C# and LabVIEW. These two folders contain samples that have been developed in the C# programming language and the LabVIEW programming language.

> **Note: The commands used in these samples are not described in this document. A detailed description can be found in the product's User's Manual.**



2 C#

The C# folder has samples that have been developed in the C# programming language. This folder has a subfolder (Mixed Devices) which contains samples that show how to write programs that work with picomotors and other supported devices. There is a solution file that can be opened in Visual Studio to see the project and the source code for each of the samples. These samples can be edited and debugged with Visual Studio Express 2008 or later, which can be downloaded for free from Microsoft's web site.



2.1 OpenMultipleDevices

The OpenMultipleDevices sample demonstrates how to communicate with several devices via USB port and/or Ethernet in one application. It uses methods in the .NET assemblies DeviceIOLib.dll and CmdLib8742.dll to perform the major steps in this sample. First it calls the constructor for DeviceIOLib and CmdLib8742. Then it initializes the I/O ports. It sets the product ID for USB device discovery to the value found in NewportUSBDriver.inf. Then it discovers devices and gets the list of device keys (one unique identifier for each discovered device). Then, for each discovered device, it displays the device key, opens the device, gets and displays its identification string, and then closes the device. After all devices have responded, communication is shut down. For more information on the methods called in this sample see section 4.

2.2 OpenSingleDevice

The OpenSingleDevice sample demonstrates how to communicate with a single device via USB port or Ethernet. It uses methods in the .NET assemblies DeviceIOLib.dll and CmdLib8742.dll to perform the major steps in this sample. First it calls the constructor for DeviceIOLib and

CmdLib8742. Then it initializes the I/O ports. It sets the product ID for USB device discovery to the value found in NewportUSBDriver.inf. Then it discovers devices and gets the first device key (a unique identifier for each discovered device). Then it opens the device, gets and displays the identification string, closes the device, and shuts down communication. For more information on the methods called in this sample see section 4.

2.3 RelativeMove

The RelativeMove sample demonstrates how to perform motion with a single device via USB or Ethernet. It uses methods in the .NET assemblies DeviceIOLib.dll and CmdLib8742.dll to perform the major steps in this sample. First it calls the constructor for DeviceIOLib and CmdLib8742. Then it initializes the I/O ports. It sets the product ID for USB device discovery to the value found in NewportUSBDriver.inf. Then it discovers devices and gets the first device key (a unique identifier for each discovered device). Then it opens the device, sets the position to zero, gets the position and displays it. Then the user is prompted for the number of steps to move. This gives the user an opportunity to exit the program without any motion occurring. Then it performs a relative move and continuously loops checking for errors, motion done, and the current position. In this loop the updated position is displayed plus any error messages that are returned by the device. After motion is done the loop is exited, the device is closed and communication is shut down.

2.4 GetPositionAllSlaves

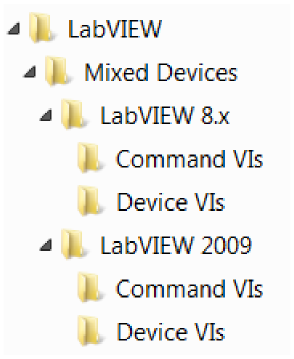
The GetPositionAllSlaves sample demonstrates how to communicate with multiple devices via USB and / or Ethernet and RS-485. It uses methods in the .NET assemblies DeviceIOLib.dll and CmdLib8742.dll to perform the major steps in this sample. First it calls the constructor for DeviceIOLib and CmdLib8742. Then it initializes the I/O ports. It sets the product ID for USB device discovery to the value found in NewportUSBDriver.inf. Then it discovers devices and gets the list of device keys (one unique identifier for each discovered device). Then, for each discovered device, it opens the master controller, gets its device address, gets its model / serial, and then displays the device key, device address, and model / serial of the master controller. Then the device addresses of the slaves that are attached to this master are retrieved. Then for each slave device, the model / serial is obtained and the device key, device address, and model / serial of the slave is displayed. Then the position of each motor is acquired and displayed. After all slaves have been queried, the master controller is closed and communication is shut down.

2.5 EnableClosedLoop

The EnableClosedLoop sample is very similar to the RelativeMove sample. The only difference is that the user is prompted to enable / disable the closed-loop setting before entering the relative steps to move. This sample demonstrates how to perform motion with a single 8743 device via USB or Ethernet. It uses methods in the .NET assemblies DeviceIOLib.dll and CmdLib8742.dll to perform the major steps in this sample. First it calls the constructor for DeviceIOLib and CmdLib8742. Then it initializes the I/O ports. It sets the product ID for USB device discovery to the value found in NewportUSBDriver.inf. Then it discovers devices and gets the first device key (a unique identifier for each discovered device). Then it opens the device, sets the position to zero, gets the position and displays it. Then the user is prompted to enable / disable the closed-loop setting and for the number of steps to move. This gives the user an opportunity to exit the program without any motion occurring. Then it performs a relative move and continuously loops checking for errors, motion done, and the current position. In this loop the updated position is displayed plus any error messages that are returned by the device. After motion is done the loop is exited, the device is closed and communication is shut down.

3 LabVIEW

The LabVIEW folder has samples that have been developed in the LabVIEW programming language. This folder has a subfolder (Mixed Devices) which contains samples that show how to write programs that work with picomotors and other supported devices. The subfolders (LabVIEW 8.x, and LabVIEW 2009) each contain the same samples written in a different version of LabVIEW.



```
LabVIEW
  Mixed Devices
    LabVIEW 8.x
      Command VIs
      Device VIs
    LabVIEW 2009
      Command VIs
      Device VIs
```

3.1 Device VIs

The Device VIs folder contains VIs that typically call a single generic I/O function (such as open, close, read, write, or query) in CmdLib8742.dll.

3.1.1 Initialize.vi

The Initialize VI must be called at the beginning of any LabVIEW program that uses DeviceIOLib or CmdLib8742. This VI performs the following steps:

1. First it calls the constructor for DeviceIOLib and CmdLib8742.
2. Then it initializes the USB port options by:
   a) setting the “IsUsingUSBEvents” flag to false, and
   b) setting the product ID for USB device discovery to the value found in NewportUSBDriver.inf.

   For more information see section 4.3.2.1.
3. Then it discovers USB and Ethernet devices followed by a 5 second delay to allow the discovery process to complete. For more information see section 4.3.3.1.
4. Then it gets the list of device keys (one unique identifier for each discovered device). For more information see section 4.3.4.

3.1.2 Shutdown.vi

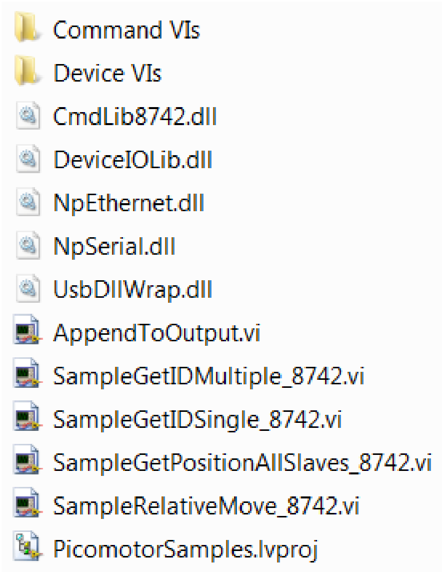
The Shutdown VI must be called at the end of any LabVIEW program that uses DeviceIOLib to cleanly shut down communication on all I/O ports.

3.2 Command VIs

The Command VIs folder contains VIs that typically call a single command in CmdLib8742.dll.

3.3 Sample Programs

The LabVIEW sample programs demonstrate how to communicate with multiple devices, how to communicate with a single device, and how to use some of the basic command VIs. The LabVIEW project file, when opened, displays a list of the sample VIs, the Command VIs, and the Device VIs along with their proper folder structure in a tree view. These VIs can be modified, executed, and debugged within the project environment. The folder that contains the LabVIEW project file must also contain the DLLs that are used by the VIs in the project.



Command VIs
Device VIs
CmdLib8742.dll
DeviceIOLib.dll
NpEthernet.dll
NpSerial.dll
UsbDllWrap.dll
AppendToOutput.vi
SampleGetIDMultiple_8742.vi
SampleGetIDSingle_8742.vi
SampleGetPositionAllSlaves_8742.vi
SampleRelativeMove_8742.vi
PicomotorSamples.lvproj

3.3.1 SampleGetIDMultiple_8742.vi

The SampleGetIDMultiple_8742.vi sample demonstrates how to communicate with several devices via USB port and/or Ethernet in one application. It uses methods in the .NET assemblies DeviceIOLib.dll and CmdLib8742.dll to perform the major steps in this sample. First it calls Initialize.vi (see section 3.1.1). This VI performs the steps needed for initialization, device discovery, and returning the list of discovered device keys. Then, for each discovered device, it calls DeviceOpen_8742.vi to open the device, calls GetMasterDeviceAddress_8742.vi to get the device address of the master controller, calls GetIdentification_8742.vi to get its identification string, adds the identification string to the Identification List indicator, and then calls DeviceClose_8742.vi to close the device. After all devices have responded, communication is shut down by calling Shutdown.vi (see section 3.1.2).

3.3.2 SampleGetIDSingle_8742.vi

The SampleGetIDSingle_8742.vi sample demonstrates how to communicate with a single device via USB port and/or serial port. It uses methods in the .NET assemblies DeviceIOLib.dll and CmdLib8742.dll to perform the major steps in this sample. First it calls Initialize.vi (see section 3.1.1). This VI performs the steps needed for initialization, device discovery, and returning the list of discovered device keys. Then it determines how many entries are in the list of device keys (one unique identifier for each discovered device). Then, if there is at least one discovered device, it calls GetFirstDeviceKey_8742.vi to get the first device key in the list calls DeviceOpen_8742.vi to open the device, calls GetMasterDeviceAddress_8742.vi to get the device address of the master controller, calls GetIdentification_8742.vi to get its identification string, displays the identification string in the Identification indicator, and then calls DeviceClose_8742.vi to close the device. Finally, communication is shut down by calling Shutdown.vi (see section 3.1.2).

3.3.3 SampleRelativeMove_8742.vi

The SampleRelativeMove_8742.vi sample demonstrates how to perform motion with a single device via USB or Ethernet. It uses methods in the .NET assemblies DeviceIOLib.dll and CmdLib8742.dll to perform the major steps in this sample. First it calls Initialize.vi (see section 3.1.1). This VI performs the steps needed for initialization, device discovery, and returning the list of discovered device keys. Then it determines how many entries are in the list of device keys (one unique identifier for each discovered device). Then, if there is at least one discovered device, it calls GetFirstDeviceKey_8742.vi to get the first device key in the list, calls DeviceOpen_8742.vi to open the device, calls GetMasterDeviceAddress_8742.vi to get the device address of the master controller, calls SetZeroPosition_8742.vi to initialize the position to zero, and calls RelativeMove_8742.vi to move the specified amount. While motion is in progress, it repeatedly calls GetErrorMsg_8742.vi to get any error message in the queue, calls GetMotionDone_8742.vi to check for motion in progress, calls GetPosition_8742.vi to get the current position and displays it, and finally aborts motion if the Stop button is pressed. After motion is done or if there is an error the loop is exited, the device is closed and communication is shut down.

3.3.4 SampleGetPositionAllSlaves_8742.vi

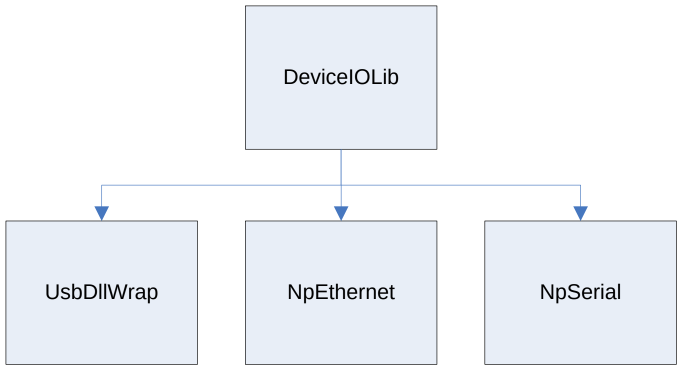
The SampleGetPositionAllSlaves_8742.vi sample demonstrates how to communicate with multiple devices via USB and / or Ethernet and RS-485. It uses methods in the .NET assemblies DeviceIOLib.dll and CmdLib8742.dll to perform the major steps in this sample. First it calls Initialize.vi (see section 3.1.1). This VI performs the steps needed for initialization, device discovery, and returning the list of discovered device keys. Then, for each discovered device, it calls DeviceOpen_8742.vi to open the device, calls GetMasterDeviceAddress_8742.vi to get the device address of the master controller, calls GetModelSerial_8742.vi to get its model / serial, and then displays the device key, device address, and model / serial of the master controller. Then it calls GetDeviceAddresses_8742.vi to get the device addresses of the slaves that are attached to this master are retrieved. Then for each slave device it calls GetModelSerial_8742.vi to get its model / serial and displays the device key, device address, and model / serial. Then it calls IsModel8743.vi to determine the number of motors (2 for 8743 and 4 for 8742). Then for each motor GetPosition_8742.vi is called to get the current position and then the motor and position is displayed. After all slaves have been queried, the master controller is closed and communication is shut down.

4 Support Libraries

DeviceIOLib.dll and CmdLib8742.dll are support libraries that work together to help a program communicate with one or more devices over various I/O ports using a common interface. These libraries contain methods (or functions) that can be called by any programming language that supports interfacing with a .NET library. Using these libraries reduces the amount of work that it takes to communicate with an instrument.

4.1 DeviceIOLib

DeviceIOLib is a library that provides a common interface for multiple instrument types and multiple I/O port types (e.g. USB, Ethernet, and serial). It sits on top of other libraries that handle a specific I/O port type (such as the Newport USB Driver). This library has methods to discover all instruments connected to a PC, to return information about discovered controllers, and to perform general device I/O (such as open, close, read, write, and query).



4.2 CmdLib8742

CmdLib8742 uses DeviceIOLib for device communication and contains all of the instrument specific methods for a 8742 or 8743 Picomotor Controller. This library has a public method for many of the commands described in the instrument user’s manual. If a particular command is not implemented in this library then a Read, Write, or Query method can be used to execute any of the commands described in the user’s manual.

4.3 Library Initialization

Initialization for the support libraries follows this general outline: call the constructor of each library to be used, specify I/O port options, indicate which I/O port types are to discover devices, perform device discovery, and return the list of device keys (one unique identifier for each discovered device).

4.3.1 Constructor

There are two DeviceIOLib constructors: one that has a logging parameter for the user to turn logging on or off, and the default constructor with no parameters where logging is false. CmdLib8742 has one constructor that accepts the DeviceIOLib object as the single parameter.

4.3.2 I/O Port Options

There are several I/O port options that may be specified. They can be divided into the following categories: USB, serial, and Ethernet.

4.3.2.1 USB Port Options

This section describes the most commonly used USB port options.

4.3.2.1.1 IsUsingUSBEvents

The IsUsingUSBEvents property determines how USB discovery will be performed. Set it to true to discover USB devices dynamically using events, otherwise set it to false to discover devices once during initialization. The default value is true. If LabVIEW is being used and this property is true, then LabVIEW must be closed and restarted each time the program is run in order to properly discover devices.

4.3.2.1.2 SetUSBProductID

The SetUSBProductID method specifies the product ID of the USB devices to be discovered. If the passed in product ID is zero then all devices will be discovered, otherwise only those devices with a matching product ID will be discovered. The default value is zero. Only those devices that have their product ID defined in NewportUSBDriver.inf will be discovered. The following table shows some of the product IDs from NewportUSBDriver.inf.

Product ID (in hex)	Device
100A	TLB-6700 Tunable Laser Controller
100D	TLB-6800 Tunable Laser Controller
4000	Picomotor Controller

4.3.2.2 Serial Port Options

This section describes the most commonly used serial port options.

4.3.2.2.1 SetSerialDiscoveryRate

The SetSerialDiscoveryRate method specifies the number of seconds that the serial port discovery process waits before repeating itself. The lower the value is, the more often the background discovery process repeats itself scanning for connected or disconnect devices. The higher the value is, the slower the response time in reporting device connection state. The valid range is from one to the maximum integer value divided by one thousand (because seconds are converted to milliseconds at a lower level). If a value outside of the valid range is specified, then it will be ignored. The default value is five seconds.

4.3.2.2.2 SetSerialPortDefaultSettings

The SetSerialPortDefaultSettings method sets the default settings for all undiscovered serial ports. The specified settings will be used when these serial ports are opened for communication.

4.3.2.2.3 GetSerialPortSettings

The GetSerialPortSettings method gets the current settings for the specified serial port.

4.3.2.2.4 SetSerialPortSettings

The SetSerialPortSettings method sets the current settings for the specified serial port. These settings will be used when the serial port is opened for communication.

4.3.2.3 Ethernet Port Options

This section describes the most commonly used Ethernet port options.

4.3.2.3.1 SetEthernetDiscoveryRate

The SetEthernetDiscoveryRate method specifies the number of seconds that the Ethernet discovery process waits before repeating itself. The lower the value is, the more often the background discovery process repeats itself scanning for connected or disconnect devices. The higher the value is, the slower the response time in reporting device connection state. The valid range is from one to the maximum integer value divided by one thousand (because seconds are converted to milliseconds at a lower level). If a value outside of the valid range is specified, then it will be ignored. The default value is five seconds.

4.3.3 Device Discovery

The device discovery process will scan for devices that are powered on and attached to the PC by the supported I/O port types. By default this process is event driven and dynamically detects changes in device connection state. The options described in section 4.3.2 above tell how to control the way this process works.

4.3.3.1 DiscoverDevices

The DiscoverDevices method is used to start the device discovery process.

The ioPortBitMask parameter allows the user to choose which I/O port types the discovery process will be performed on (e.g. USB, serial, and / or Ethernet). Bit #0 is USB, bit #1 is serial, and bit #2 is Ethernet. If a bit is on then discovery will be performed for that I/O port type, if a bit is set to zero then discovery will not be performed for that I/O port type. Each bit has an integer value of 2 to the n^{th} power, where n is the bit number (e.g. USB = 2^0 = 1, serial = 2^1 = 2, and Ethernet = 2^2 = 4). These values can be added together to discover multiple I/O port types.

The msecDelayForDiscovery parameter is useful when the DeviceDiscoveryChanged event is not being used for dynamic device discovery. The number of milliseconds to delay indicates how long the method will delay after starting the device discovery process. This delay allows the device discovery process enough time to completely discover all devices that are currently connected so that a one-time snapshot of discovered devices (device keys) can be created before the user attempts any device communication. If Ethernet is being used then the delay should be around 5000 milliseconds, otherwise it should be around 1500 milliseconds.

4.3.3.2 DeviceDiscoveryChanged

The DeviceDiscoveryChanged event is fired by DeviceIOLib when it detects that a device connection state has changed. The DiscoveryEventArgs specify whether the device is being added to or removed from the list of discovered devices. If a program handles this event, then it can dynamically update its user interface to show which devices are currently available for communication. When this event is fired, DeviceIOLib has already updated its list of device keys (one unique identifier for each discovered device).

4.3.3.3 GetPortType

The GetPortType method returns the I/O port type (USB, serial, or Ethernet) for the specified device. This method could be used in a DeviceDiscoveryChanged event handler if different handling is needed based upon the I/O port type.

4.3.3.4 GetPortObject

The GetPortObject method returns the underlying I/O port communication object. This object can be used to directly access UsbDllWrap.dll, NpSerial.dll, or NpEthernet.dll. This method can be used to get the USB object that is passed into a constructor of PowerMeterCommands.dll if a power meter needed to be used in the same program.

4.3.4 Initialization Result

If all goes well during the steps of library initialization (described in section 4.3), then the result will be (1) an object reference to each library that had its constructor called, and (2) a list of device keys (one unique identifier for each discovered device). A device key is used in many of the library methods to specify which device to communicate with. Any library method that accepts a device key as a parameter will only work properly with a device key that is currently in the list of discovered devices.

4.3.4.1 GetDeviceCount

The GetDeviceCount method returns the number of discovered devices.

4.3.4.2 GetFirstDeviceKey

The GetFirstDeviceKey method returns the first device key from the list of discovered devices. Serial ports are given lowest priority.

4.3.4.3 GetDeviceKeys

The GetDeviceKeys method returns all the device keys from the list of discovered devices.

4.3.4.4 GetDeviceKeysExclude

The GetDeviceKeysExclude method returns the device keys from the list of discovered devices, except for those in the exclude list. The “exclude” parameter is a list of device keys to exclude from the returned list.

4.3.4.5 GetDeviceKeysFilter

The GetDeviceKeysFilter method returns the device keys from the list of discovered devices that match the specified filter. The “filter” parameter is a list of models to include in the returned list.

4.4 Library Shutdown

Before a program exits, all open devices should be closed and the Shutdown method in DeviceIOLib should be called to properly clean up any communication background tasks.

4.5 Logging

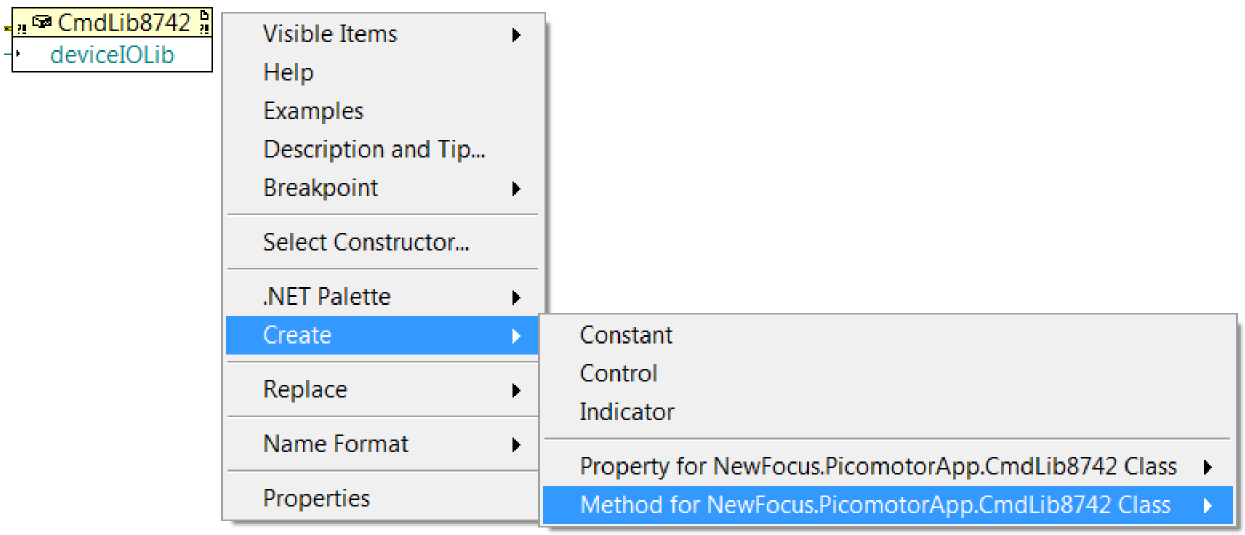
Logging can be turned on by passing a Boolean true value into the DeviceIOLib constructor. This can be useful when trying to determine the cause of communication errors or to see the data that is being transferred between the PC and the device. When logging is turned on the following information is written to the log file: general information about the operating system, device discovery information, data being written to the device, response data being read from the device, I/O error codes, event name and state information, and general debugging information. If a C# exception is thrown then this information is logged even if logging is not turned on. Log files are named Log<YYYY-MM-DD>.txt and are created in one of the following folders (depending upon system settings and permissions): (1) the \Log folder located in the directory containing the current executable (.exe), (2) the same folder path as in #1, except the highest level folder is replaced with “My Documents” (e.g. C:\Program Files\Newport\Install

Folder\Bin\Log would be replaced with C:\My Documents\Newport\Install Folder\Bin\Log), and (3) the My Documents\Log folder.

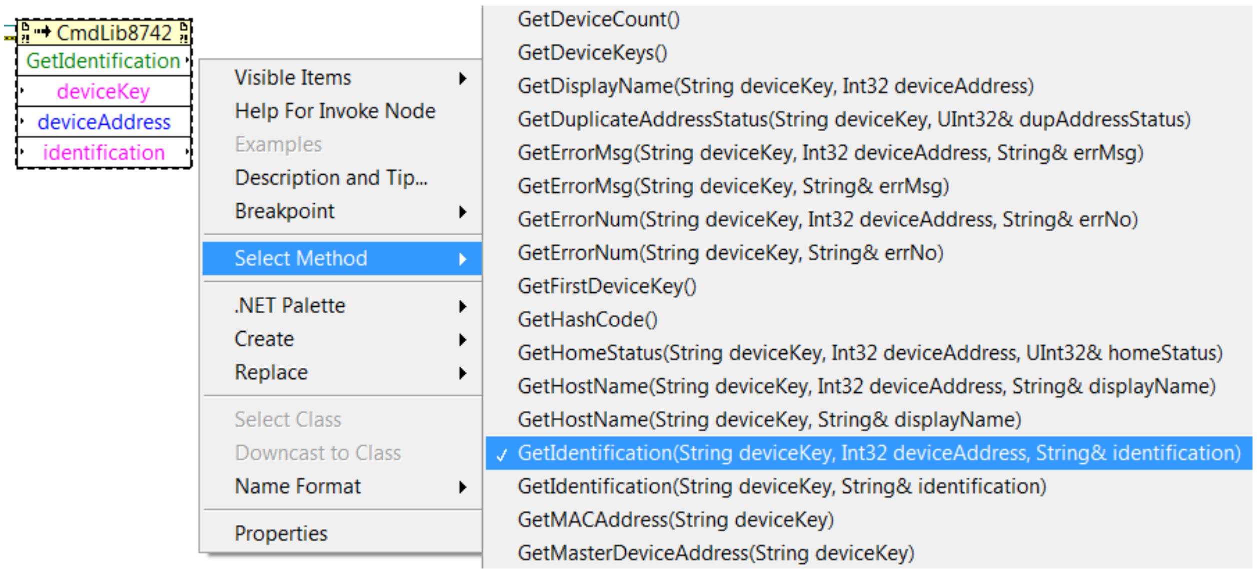
4.6 Support Libraries in LabVIEW

Since LabVIEW supports calling methods in a .NET library, any of the constructors and methods in DeviceIOLib.dll and CmdLib8742.dll can be directly called by a LabVIEW VI. Many of these methods have been wrapped in a VI so that a VI can be called instead. These wrapper VIs just call a single method in the library. The Command VIs folder contains wrapper VIs for many of the commands described in the user's manual. If a wrapper VI does not exist for a particular command then a method in the library may exist to perform this command. If neither exist, then a Read, Write, or Query method in the library can be used to perform any of the commands described in the user's manual. The Device VIs folder contains wrapper VIs that call some of these general I/O methods (such as open, close, read, write, and query).

As stated in section 4 above, there are only a few basic steps required in order to communicate with an instrument. The first step is to initialize device communication. This can be done by calling Initialize.vi (from the Device VIs folder). The design of Initialize.vi is described in section 4.3. The second step is to select a device key from the DeviceKey List returned by Initialize.vi or call GetFirstDeviceKey (described in section 4.3.4.2). The third step is to send a command or query to the instrument. This can be done by calling a VI from the Command VIs folder or by directly calling a method from the library. To call a method in the library, simply right-click the upper right hand corner of any DeviceIOLib or CmdLib8742 block (on the terminal or library reference wire) to display a context menu and select "Create". Then select "Method for NewFocus.PicomotorApp.CmdLib8742 Class", and select the name of the method to be called from the list in the context menu.



If a LabVIEW Invoke Node (that specifies a library method) has been copied and pasted into a VI it can be modified by right-clicking the block, selecting “Select Method” from the context menu, and then selecting a method from the list in the context menu. If the data type changes on any of the inputs or outputs then this part will have to be rewired with the correct data type.


CmdLib8742
GetIdentification
deviceKey
deviceAddress
identification
Visible Items
Help For Invoke Node
Examples
Description and Tip...
Breakpoint
Select Method
.NET Palette
Create
Replace
Select Class
Downcast to Class
Name Format
Properties
GetDeviceCount()
GetDeviceKeys()
GetDisplayName(String deviceKey, Int32 deviceAddress)
GetDuplicateAddressStatus(String deviceKey, UInt32& dupAddressStatus)
GetErrorMsg(String deviceKey, Int32 deviceAddress, String& errMsg)
GetErrorMsg(String deviceKey, String& errMsg)
GetErrorNum(String deviceKey, Int32 deviceAddress, String& errNo)
GetErrorNum(String deviceKey, String& errNo)
GetFirstDeviceKey()
GetHashCode()
GetHomeStatus(String deviceKey, Int32 deviceAddress, UInt32& homeStatus)
GetHostName(String deviceKey, Int32 deviceAddress, String& displayName)
GetHostName(String deviceKey, String& displayName)
✓ GetIdentification(String deviceKey, Int32 deviceAddress, String& identification)
GetIdentification(String deviceKey, String& identification)
GetMACAddress(String deviceKey)
GetMasterDeviceAddress(String deviceKey)

The final step is to call the Shutdown method in DeviceIOLib to clean up background communication tasks.

4.7 The DeviceIOLib API

This section describes the public interface for DeviceIOLib.dll. The following constructors and methods are available to any programming language that supports calling a function within a .NET library.

```
/// <summary>
/// The event that is fired when a DeviceDiscoveryChanged event occurs.
/// </summary>
public event DeviceDiscoveryDelegate DeviceDiscoveryChanged;

/// <summary>
/// The maximum string length for an I/O transfer.
/// </summary>
public const int m_kMaxXferLen = 64;

/// <summary>
/// Constructor.
/// </summary>
public DeviceIOLib ()

/// <summary>
/// Constructor.
/// </summary>
/// <param name="isLogging">True to turn on logging, otherwise false.</param>
public DeviceIOLib (bool isLogging)

/// <summary>
/// This property gets the logging flag.
/// </summary>
public virtual bool IsLogging

/// <summary>
/// This property gets / sets the flag that determines how USB device discovery is
/// performed.
/// True to discover USB devices dynamically using events, otherwise false to discover
/// once during initialization.
/// </summary>
public virtual bool IsUsingUSBEvents

/// <summary>
/// This method sets the product ID to use as a filter for devices to open.
/// </summary>
/// <param name="productID">The product ID.</param>
public virtual void SetUSBProductID (int productID)

/// <summary>
/// This method gets the MAC address of the specified device if it is in the list of
/// discovered devices.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>The MAC address of the discovered device, otherwise an empty
/// string.</returns>
public virtual string GetMACAddress (string deviceKey)

/// <summary>
/// This method sets the rate at which the Ethernet discovery process repeats itself.
/// </summary>
/// <param name="seconds">The number of seconds.</param>
public virtual void SetEthernetDiscoveryRate (int seconds)

/// <summary>
/// This method sets the rate at which the serial port discovery process repeats itself.
/// </summary>
/// <param name="seconds">The number of seconds.</param>
public virtual void SetSerialDiscoveryRate (int seconds)

/// <summary>
/// This method gets the current default settings for the serial port communication
/// object.
/// </summary>
/// <param name="baudRate">The baud rate.</param>
/// <param name="parity">The parity.</param>
/// <param name="dataBits">The number of data bits.</param>
/// <param name="stopBits">The number of stop bits.</param>
/// <param name="handshake">The handshake protocol.</param>
/// <param name="readTimeout">The read timeout in milliseconds.</param>
/// <param name="writeTimeout">The write timeout in milliseconds.</param>
/// <param name="newLine">The new line sequence for reads and writes.</param>
public virtual void GetSerialPortDefaultSettings (ref int baudRate, ref Parity parity,
   ref int dataBits, ref StopBits stopBits, ref Handshake handshake, ref int readTimeout,
   ref int writeTimeout, ref string newLine)

/// <summary>
/// This method sets the current default settings for the serial port communication
/// object.
/// </summary>
/// <param name="baudRate">The baud rate.</param>
/// <param name="parity">The parity.</param>
/// <param name="dataBits">The number of data bits.</param>
/// <param name="stopBits">The number of stop bits.</param>
/// <param name="handshake">The handshake protocol.</param>
/// <param name="readTimeout">The read timeout in milliseconds.</param>
/// <param name="writeTimeout">The write timeout in milliseconds.</param>
/// <param name="newLine">The new line sequence for reads and writes.</param>
public virtual void SetSerialPortDefaultSettings (int baudRate, Parity parity, int
   dataBits, StopBits stopBits, Handshake handshake, int readTimeout, int writeTimeout,
   string newLine)

/// <summary>
/// This method gets the current settings for the specified serial port.
/// </summary>
/// <param name="deviceKey">The device key of the serial port.</param>
/// <param name="baudRate">The baud rate.</param>
/// <param name="parity">The parity.</param>
/// <param name="dataBits">The number of data bits.</param>
/// <param name="stopBits">The number of stop bits.</param>
/// <param name="handshake">The handshake protocol.</param>
/// <param name="readTimeout">The read timeout in milliseconds.</param>
/// <param name="writeTimeout">The write timeout in milliseconds.</param>
/// <param name="newLine">The new line sequence for reads and writes.</param>
/// <returns>True if the serial port settings could be retrieved, otherwise
/// false.</returns>
public virtual bool GetSerialPortSettings (string deviceKey, ref int baudRate, ref Parity
   parity, ref int dataBits, ref StopBits stopBits, ref Handshake handshake, ref int
   readTimeout, ref int writeTimeout, ref string newLine)

/// <summary>
/// This method sets the current settings for the specified serial port.
/// </summary>
/// <param name="deviceKey">The device key of the serial port.</param>
/// <param name="baudRate">The baud rate.</param>
/// <param name="parity">The parity.</param>
/// <param name="dataBits">The number of data bits.</param>
/// <param name="stopBits">The number of stop bits.</param>
/// <param name="handshake">The handshake protocol.</param>
/// <param name="readTimeout">The read timeout in milliseconds.</param>
/// <param name="writeTimeout">The write timeout in milliseconds.</param>
/// <param name="newLine">The new line sequence for reads and writes.</param>
/// <returns>True if the serial port settings could be retrieved, otherwise
/// false.</returns>
public virtual bool SetSerialPortSettings (string deviceKey, int baudRate, Parity parity,
   int dataBits, StopBits stopBits, Handshake handshake, int readTimeout, int
   writeTimeout, string newLine)

/// <summary>
/// This method gets the port type for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>The port type.</returns>
public virtual ePortType GetPortType (string deviceKey)

/// <summary>
/// This method gets the underlying port object of the specified port type.
/// </summary>
/// <param name="portType">The port type.</param>
/// <returns>The underlying port object or null for an error.</returns>
public virtual object GetPortObject (ePortType portType)

/// <summary>
/// This method discovers the devices that are available for communication.
/// </summary>
/// <param name="ioPortBitMask">The I/O Port bit mask to specify which I/O ports to
/// perform
/// discovery (Bit 0 = USB, Bit 1 = Serial, Bit 2 = Ethernet).</param>
/// <param name="msecDelayForDiscovery">The number of milliseconds to wait for devices to
/// be discovered.</param>
public virtual void DiscoverDevices (uint ioPortBitMask, int msecDelayForDiscovery)

/// <summary>
/// This method rediscovers the specified device at the application level (the
/// underlying DLLs for specific I/O port types are not affected).
/// </summary>
/// <param name="deviceKey">The device key.</param>
public virtual void RediscoverDevice (string deviceKey)

/// <summary>
/// This method gets the number of discovered devices.
/// </summary>
/// <returns>The number of discovered devices.</returns>
public virtual int GetDeviceCount ()
```

```
/// <summary>
/// This method gets the first device key from the list of discovered devices.
/// </summary>
/// <returns>The first device key from the list of discovered devices.</returns>
public virtual string GetFirstDeviceKey ()

/// <summary>
/// This method gets all the device keys from the list of discovered devices.
/// </summary>
/// <returns>All the device keys that have been discovered.</returns>
public virtual string[] GetDeviceKeys ()

/// <summary>
/// This method gets the device keys from the list of discovered devices,
/// except for those in the "exclude" list.
/// </summary>
/// <param name="exclude">The list of device keys to exclude from the returned
/// list.</param>
/// <returns>The device keys that have been discovered minus those in the "exclude"
/// list.</returns>
public virtual string[] GetDeviceKeysExclude (string[] exclude)

/// <summary>
/// This method gets the device keys from the list of discovered devices that match the
/// specified filter.
/// </summary>
/// <param name="filter">The list of models to include in the returned list.</param>
/// <returns>The filtered list of device keys that have been discovered.</returns>
public virtual string[] GetDeviceKeysFilter (string[] filter)

/// <summary>
/// This method determines if the passed in device key is valid or not.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>True if the device key is in the list of discovered devices, otherwise
/// false.</returns>
public virtual bool DeviceKeyIsValid (string deviceKey)

/// <summary>
/// This method gets the model and serial number from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>The model and serial number separated by a blank.</returns>
public virtual string GetModelSerial (string deviceKey)

/// <summary>
/// This method sets the model and serial number for the specified device by calling
/// 'IdentifyInstrument'.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool SetModelSerial (string deviceKey)

/// <summary>
/// This method gets the display name from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>The display name.</returns>
public virtual string GetDisplayName (string deviceKey)

/// <summary>
/// This method sets the display name for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="displayName">The display name.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool SetDisplayName (string deviceKey, string displayName)

/// <summary>
/// This method identifies the instrument by sending the *IDN? query and parsing the
/// response data.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="model">The model.</param>
/// <param name="serialNum">The serial number.</param>
/// <param name="fwVersion">The firmware version.</param>
/// <param name="fwDate">The firmware date.</param>
public virtual void IdentifyInstrument (string deviceKey, out string model, out string
   serialNum, out string fwVersion, out string fwDate)

/// <summary>
/// This method parses the information from the instrument identification string.
/// </summary>
/// <param name="identification">The instrument identification string.</param>
/// <param name="model">The model.</param>
/// <param name="serialNum">The serial number.</param>
/// <param name="fwVersion">The firmware version.</param>
/// <param name="fwDate">The firmware date.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool ParseIDInfo (string identification, out string model, out string
   serialNum, out string fwVersion, out string fwDate)

/// <summary>
/// This method parses the model and serial number from the device key.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="model">The model.</param>
/// <param name="serialNum">The serial number.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool ParseModelSerial (string deviceKey, ref string model, ref string
   serialNum)

/// <summary>
/// This method opens the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool Open (string deviceKey)

/// <summary>
/// This method closes the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool Close (string deviceKey)

/// <summary>
/// This method reads a string response from the device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="value">The string response.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool Read (string deviceKey, ref string value)

/// <summary>
/// This method reads data from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="buffer">The buffer to hold the response data.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool Read (string deviceKey, StringBuilder buffer)

/// <summary>
/// This method reads data from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="buffer">The buffer to hold the response data.</param>
/// <param name="nNumBytesRead">The number of bytes read.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool Read (string deviceKey, byte[] buffer, out int nNumBytesRead)
```

```
/// <summary>
/// This method writes data to the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="buffer">The buffer to hold the data.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool Write (string deviceKey, string buffer)

/// <summary>
/// This method writes data to the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="buffer">The buffer to hold the data.</param>
/// <param name="dataLength">The length of the data in the buffer.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool Write (string deviceKey, byte[] buffer, int dataLength)

/// <summary>
/// This method sends the passed in query to the device and returns the string response.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The query string.</param>
/// <param name="value">The string response.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool Query (string deviceKey, string cmd, ref string value)

/// <summary>
/// This method sends the passed in command string to the specified device
/// and reads the response data.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="command">The command string to send.</param>
/// <param name="buffer">The buffer to hold the response data.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool Query (string deviceKey, string command, StringBuilder buffer)

/// <summary>
/// This method writes the passed in string to the log file if logging is turned on.
/// </summary>
/// <param name="outputText">The text to output.</param>
public virtual void WriteLog (string outputText)

/// <summary>
/// This method writes the passed in arguments to the log file according
/// to the specified format.
/// </summary>
/// <param name="logging">True if logging, otherwise false.</param>
/// <param name="format">The format specifier.</param>
/// <param name="args">The data values to output.</param>
public virtual void WriteLog (bool logging, string format, params object[] args)

/// <summary>
/// This method writes the passed in arguments to the log file according
/// to the specified format.
/// </summary>
/// <param name="format">The format specifier.</param>
/// <param name="args">The data values to output.</param>
public virtual void WriteLog (string format, params object[] args)

/// <summary>
/// This method reads a string response from the device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="value">The string response.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool ReadString (string deviceKey, ref string value)

/// <summary>
/// This method sends the passed in command to the device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The command string.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool WriteString (string deviceKey, string cmd)

/// <summary>
/// This method sends the passed in query to the device and returns the string response.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The query string.</param>
/// <param name="value">The string response.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool QueryString (string deviceKey, string cmd, ref string value)

/// <summary>
/// This method sends the passed in command to the device along with an integer
/// parameter.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The command string.</param>
/// <param name="value">The integer parameter.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool WriteInt (string deviceKey, string cmd, int value)

/// <summary>
/// This method sends the passed in query to the device and returns the integer response.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The query string.</param>
/// <param name="value">The integer response.</param>
/// <param name="shouldConvert">True if the response string should be converted to a
/// number, otherwise false.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool QueryInt (string deviceKey, string cmd, ref int value, bool
   shouldConvert)

/// <summary>
/// This method sends the passed in command to the device along with an unsigned integer
/// parameter.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The command string.</param>
/// <param name="value">The unsigned integer parameter.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool WriteUInt (string deviceKey, string cmd, uint value)

/// <summary>
/// This method sends the passed in query to the device and returns the unsigned integer
/// response.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The query string.</param>
/// <param name="value">The unsigned integer response.</param>
/// <param name="shouldConvert">True if the response string should be converted to a
/// number, otherwise false.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool QueryUInt (string deviceKey, string cmd, ref uint value, bool
   shouldConvert)

/// <summary>
/// This method sends the passed in command to the device along with a floating point
/// parameter.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The command string.</param>
/// <param name="value">The floating point parameter.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool WriteFloat (string deviceKey, string cmd, float value)

/// <summary>
/// This method sends the passed in query to the device and returns the floating point
/// response.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The query string.</param>
/// <param name="value">The floating point response.</param>
/// <param name="shouldConvert">True if the response string should be converted to a
/// number, otherwise false.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool QueryFloat (string deviceKey, string cmd, ref float value, bool
   shouldConvert)

/// <summary>
/// This method sends the passed in command to the device along with a decimal parameter.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The command string.</param>
/// <param name="value">The decimal parameter.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool WriteDecimal (string deviceKey, string cmd, decimal value)

/// <summary>
/// This method sends the passed in query to the device and returns the decimal response.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The query string.</param>
/// <param name="value">The decimal response.</param>
/// <param name="shouldConvert">True if the response string should be converted to a
/// number, otherwise false.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool QueryDecimal (string deviceKey, string cmd, ref decimal value, bool
   shouldConvert)

/// <summary>
/// This method sends the passed in command to the device along with a double-precision
/// parameter.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The command string.</param>
/// <param name="value">The double-precision parameter.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool WriteDouble (string deviceKey, string cmd, double value)

/// <summary>
/// This method sends the passed in query to the device and returns the double-precision
/// response.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The query string.</param>
/// <param name="value">The double-precision response.</param>
/// <param name="shouldConvert">True if the response string should be converted to a
/// number, otherwise false.</param>
/// <returns>True for success, false for failure.</returns>
public virtual bool QueryDouble (string deviceKey, string cmd, ref double value, bool
   shouldConvert)

/// <summary>
/// This method performs the required steps to cleanly stop communication.
/// </summary>
public virtual void Shutdown ()
```

4.8 The CmdLib8742 API

This section describes the public interface for CmdLib8742.dll. The following constructors and methods are available to any programming language that supports calling a function within a .NET library.

```
/// <summary>
/// Constructor.
/// </summary>
public CmdLib8742 (DeviceIOLib deviceIOLib)

/// <summary>
/// This method handles the DeviceDiscoveryChanged event.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="e">The event arguments.</param>
private void OnDeviceDiscoveryChanged (string deviceKey, DiscoveryEventArgs e)

/// <summary>
/// This method gets the identification string from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="identification">The identification string.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetIdentification (string deviceKey, ref string identification)

/// <summary>
/// This method gets the identification string from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="identification">The identification string.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetIdentification (string deviceKey, int deviceAddress, ref string
   identification)

/// <summary>
/// This method gets the host name from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="displayName">The host name.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetHostName (string deviceKey, ref string displayName)

/// <summary>
/// This method gets the host name from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="displayName">The host name.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetHostName (string deviceKey, int deviceAddress, ref string displayName)

/// <summary>
/// This method sets the host name for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="displayName">The host name (16 char. max.).</param>
/// <returns>True for success, false for failure.</returns>
public bool SetHostName (string deviceKey, string displayName)

/// <summary>
/// This method sets the host name for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="displayName">The host name (16 char. max.).</param>
/// <returns>True for success, false for failure.</returns>
public bool SetHostName (string deviceKey, int deviceAddress, string displayName)
```

```
/// <summary>
/// This method gets an error number from the error queue of the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="errNo">The error number.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetErrorNum (string deviceKey, ref string errNo)

/// <summary>
/// This method gets an error number from the error queue of the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="errNo">The error number.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetErrorNum (string deviceKey, int deviceAddress, ref string errNo)

/// <summary>
/// This method gets an error message from the error queue of the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="errMsg">The error message.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetErrorMsg (string deviceKey, ref string errMsg)

/// <summary>
/// This method gets an error message from the error queue of the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="errMsg">The error message.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetErrorMsg (string deviceKey, int deviceAddress, ref string errMsg)

/// <summary>
/// This method performs a motor check on the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>True for success, false for failure.</returns>
public bool MotorCheck (string deviceKey)

/// <summary>
/// This method performs a motor check on the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <returns>True for success, false for failure.</returns>
public bool MotorCheck (string deviceKey, int deviceAddress)

/// <summary>
/// This method gets the motor type from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="motorType">The motor type (0 = No Motor, 1 = Undefined, 2 = Tiny, 3 =
/// Standard).</param>
/// <returns>True for success, false for failure.</returns>
public bool GetMotorType (string deviceKey, int motor, ref eMotorType motorType)

/// <summary>
/// This method gets the motor type from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="motorType">The motor type (0 = No Motor, 1 = Undefined, 2 = Tiny, 3 =
/// Standard).</param>
/// <returns>True for success, false for failure.</returns>
public bool GetMotorType (string deviceKey, int deviceAddress, int motor, ref eMotorType
   motorType)
```

```
/// <summary>
/// This method sets the motor type for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="motorType">The motor type (0 = No Motor, 1 = Undefined, 2 = Tiny, 3 =
/// Standard).</param>
/// <returns>True for success, false for failure.</returns>
public bool SetMotorType (string deviceKey, int motor, eMotorType motorType)

/// <summary>
/// This method sets the motor type for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="motorType">The motor type (0 = No Motor, 1 = Undefined, 2 = Tiny, 3 =
/// Standard).</param>
/// <returns>True for success, false for failure.</returns>
public bool SetMotorType (string deviceKey, int deviceAddress, int motor, eMotorType
   motorType)

/// <summary>
/// This method gets the current position from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="position">The current position.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetPosition (string deviceKey, int motor, ref int position)

/// <summary>
/// This method gets the current position from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="position">The current position.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetPosition (string deviceKey, int deviceAddress, int motor, ref int
   position)

/// <summary>
/// This method gets the relative steps setting from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="relativeSteps">The relative steps setting.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetRelativeSteps (string deviceKey, int motor, ref int relativeSteps)

/// <summary>
/// This method gets the relative steps setting from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="relativeSteps">The relative steps setting.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetRelativeSteps (string deviceKey, int deviceAddress, int motor, ref int
   relativeSteps)

/// <summary>
/// This method performs a relative move on the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="relativeSteps">The relative steps to move.</param>
/// <returns>True for success, false for failure.</returns>
public bool RelativeMove (string deviceKey, int motor, int relativeSteps)
```

```
/// <summary>
/// This method performs a relative move on the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="relativeSteps">The relative steps to move.</param>
/// <returns>True for success, false for failure.</returns>
public bool RelativeMove (string deviceKey, int deviceAddress, int motor, int
   relativeSteps)

/// <summary>
/// This method gets the absolute target position from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="targetPos">The absolute target position.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetAbsTargetPos (string deviceKey, int motor, ref int targetPos)

/// <summary>
/// This method gets the absolute target position from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="targetPos">The absolute target position.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetAbsTargetPos (string deviceKey, int deviceAddress, int motor, ref int
   targetPos)

/// <summary>
/// This method performs an absolute move on the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="targetPos">The absolute target position to move to.</param>
/// <returns>True for success, false for failure.</returns>
public bool AbsoluteMove (string deviceKey, int motor, int targetPos)

/// <summary>
/// This method performs an absolute move on the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="targetPos">The absolute target position to move to.</param>
/// <returns>True for success, false for failure.</returns>
public bool AbsoluteMove (string deviceKey, int deviceAddress, int motor, int targetPos)

/// <summary>
/// This method performs a jog in the negative direction on the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <returns>True for success, false for failure.</returns>
public bool JogNegative (string deviceKey, int motor)

/// <summary>
/// This method performs a jog in the negative direction on the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <returns>True for success, false for failure.</returns>
public bool JogNegative (string deviceKey, int deviceAddress, int motor)

/// <summary>
/// This method performs a jog in the positive direction on the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <returns>True for success, false for failure.</returns>
public bool JogPositive (string deviceKey, int motor)

/// <summary>
/// This method performs a jog in the positive direction on the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <returns>True for success, false for failure.</returns>
public bool JogPositive (string deviceKey, int deviceAddress, int motor)

/// <summary>
/// This method performs an abort motion on the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>True for success, false for failure.</returns>
public bool AbortMotion (string deviceKey)

/// <summary>
/// This method performs an abort motion on the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <returns>True for success, false for failure.</returns>
public bool AbortMotion (string deviceKey, int deviceAddress)

/// <summary>
/// This method performs a stop motion on the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <returns>True for success, false for failure.</returns>
public bool StopMotion (string deviceKey, int motor)

/// <summary>
/// This method performs a stop motion on the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <returns>True for success, false for failure.</returns>
public bool StopMotion (string deviceKey, int deviceAddress, int motor)

/// <summary>
/// This method sets the zero position (define home) for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <returns>True for success, false for failure.</returns>
public bool SetZeroPosition (string deviceKey, int motor)

/// <summary>
/// This method sets the zero position (define home) for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <returns>True for success, false for failure.</returns>
public bool SetZeroPosition (string deviceKey, int deviceAddress, int motor)

/// <summary>
/// This method gets the motion done status from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="isMotionDone">The motion done status.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetMotionDone (string deviceKey, int motor, ref bool isMotionDone)
```

```
/// <summary>
/// This method gets the motion done status from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="isMotionDone">The motion done status.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetMotionDone (string deviceKey, int deviceAddress, int motor, ref bool
   isMotionDone)

/// <summary>
/// This method gets the velocity from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="stepsPerSec">The velocity in steps per second.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetVelocity (string deviceKey, int motor, ref int stepsPerSec)

/// <summary>
/// This method gets the velocity from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="stepsPerSec">The velocity in steps per second.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetVelocity (string deviceKey, int deviceAddress, int motor, ref int
   stepsPerSec)

/// <summary>
/// This method sets the velocity for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="stepsPerSec">The velocity in steps per second.</param>
/// <returns>True for success, false for failure.</returns>
public bool SetVelocity (string deviceKey, int motor, int stepsPerSec)

/// <summary>
/// This method sets the velocity for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="stepsPerSec">The velocity in steps per second.</param>
/// <returns>True for success, false for failure.</returns>
public bool SetVelocity (string deviceKey, int deviceAddress, int motor, int stepsPerSec)

/// <summary>
/// This method gets the acceleration from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="stepsPerSec2">The acceleration in steps per second squared.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetAcceleration (string deviceKey, int motor, ref int stepsPerSec2)

/// <summary>
/// This method gets the acceleration from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="stepsPerSec2">The acceleration in steps per second squared.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetAcceleration (string deviceKey, int deviceAddress, int motor, ref int
   stepsPerSec2)

/// <summary>
/// This method sets the acceleration for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="stepsPerSec2">The acceleration in steps per second squared.</param>
/// <returns>True for success, false for failure.</returns>
public bool SetAcceleration (string deviceKey, int motor, int stepsPerSec2)

/// <summary>
/// This method sets the acceleration for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="stepsPerSec2">The acceleration in steps per second squared.</param>
/// <returns>True for success, false for failure.</returns>
public bool SetAcceleration (string deviceKey, int deviceAddress, int motor, int
   stepsPerSec2)

/// <summary>
/// This method saves settings in the device's volatile memory to its persistent memory.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>True for success, false for failure.</returns>
public bool SaveToMemory (string deviceKey)

/// <summary>
/// This method saves settings in the device's volatile memory to its persistent memory.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <returns>True for success, false for failure.</returns>
public bool SaveToMemory (string deviceKey, int deviceAddress)

/// <summary>
/// This method gets the scan status word from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="scanStatus">The scan status word.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetScanStatus (string deviceKey, ref uint scanStatus)

/// <summary>
/// This method scans a master controller for slave controllers.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="option">The scan option (0 = None, 1 = Preserve, 2 = Reset All).</param>
/// <returns>True for success, false for failure.</returns>
public bool Scan (string deviceKey, int option)

/// <summary>
/// This method gets the scan done status from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="isScanDone">The scan done status.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetScanDone (string deviceKey, ref bool isScanDone)

/// <summary>
/// This method gets the device address from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetDeviceAddress (string deviceKey, ref int deviceAddress)

/// <summary>
/// This method sets the device address of the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The current device address.</param>
/// <param name="newDeviceAddress">The new device address (to be set to).</param>
/// <returns>True for success, false for failure.</returns>
public bool SetDeviceAddress (string deviceKey, int deviceAddress, int newDeviceAddress)

/// <summary>
/// This method gets the duplicate address status word from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="dupAddressStatus">The duplicate address status word.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetDuplicateAddressStatus (string deviceKey, ref uint dupAddressStatus)

/// <summary>
/// This method resets a master controller so that it can be rediscovered.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>True for success, false for failure.</returns>
public bool Reset (string deviceKey)

/// <summary>
/// This method gets the closed-loop enabled setting from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="setting">The closed-loop enabled setting (0 = disabled, 1 =
/// enabled).</param>
/// <returns>True for success, false for failure.</returns>
public bool GetCLEnabledSetting (string deviceKey, int deviceAddress, int motor, ref int
   setting)

/// <summary>
/// This method sets the closed-loop enabled setting for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="setting">The closed-loop enabled setting (0 = disabled, 1 =
/// enabled).</param>
/// <returns>True for success, false for failure.</returns>
public bool SetCLEnabledSetting (string deviceKey, int deviceAddress, int motor, int
   setting)

/// <summary>
/// This method performs a Move To Travel Limit in the positive direction on the
/// specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <returns>True for success, false for failure.</returns>
public bool MoveToTravelLimitPos (string deviceKey, int deviceAddress, int motor)

/// <summary>
/// This method performs a Move To Travel Limit in the negative direction on the
/// specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <returns>True for success, false for failure.</returns>
public bool MoveToTravelLimitNeg (string deviceKey, int deviceAddress, int motor)

/// <summary>
/// This method performs a Move To Next - Dir Index on the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <returns>True for success, false for failure.</returns>
public bool MoveToNextDirIndexNeg (string deviceKey, int deviceAddress, int motor)
```

```
/// <summary>
/// This method performs a Move To Next + Dir Index on the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <returns>True for success, false for failure.</returns>
public bool MoveToNextDirIndexPos (string deviceKey, int deviceAddress, int motor)

/// <summary>
/// This method gets the home status word from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="homeStatus">The home status word.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetHomeStatus (string deviceKey, int deviceAddress, ref uint homeStatus)

/// <summary>
/// This method gets the closed-loop units from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="units">The closed-loop units (0 = steps, 1 = counts).</param>
/// <returns>True for success, false for failure.</returns>
public bool GetCLUnits (string deviceKey, int deviceAddress, int motor, ref int units)

/// <summary>
/// This method sets the closed-loop units for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="units">The closed-loop units (0 = steps, 1 = counts).</param>
/// <returns>True for success, false for failure.</returns>
public bool SetCLUnits (string deviceKey, int deviceAddress, int motor, int units)

/// <summary>
/// This method gets the closed-loop step resolution from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="stepResolution">The step resolution in counts per step.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetCLStepResolution (string deviceKey, int deviceAddress, int motor, ref
   float stepResolution)

/// <summary>
/// This method sets the closed-loop step resolution for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="stepResolution">The step resolution in counts per step.</param>
/// <returns>True for success, false for failure.</returns>
public bool SetCLStepResolution (string deviceKey, int deviceAddress, int motor, float
   stepResolution)

/// <summary>
/// This method gets the closed-loop deadband from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="deadband">The deadband in counts.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetCLDeadband (string deviceKey, int deviceAddress, int motor, ref int
   deadband)
```

```
/// <summary>
/// This method sets the closed-loop deadband for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="deadband">The deadband in counts.</param>
/// <returns>True for success, false for failure.</returns>
public bool SetCLDeadband (string deviceKey, int deviceAddress, int motor, int deadband)

/// <summary>
/// This method gets the closed-loop update interval from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="updateInterval">The update interval in seconds.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetCLUpdateInterval (string deviceKey, int deviceAddress, int motor, ref
   float updateInterval)

/// <summary>
/// This method sets the closed-loop update interval for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="updateInterval">The update interval in seconds.</param>
/// <returns>True for success, false for failure.</returns>
public bool SetCLUpdateInterval (string deviceKey, int deviceAddress, int motor, float
   updateInterval)

/// <summary>
/// This method gets the closed-loop following error threshold from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="threshold">The following error threshold in counts.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetCLThreshold (string deviceKey, int deviceAddress, int motor, ref int
   threshold)

/// <summary>
/// This method sets the closed-loop following error threshold for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="threshold">The following error threshold in counts.</param>
/// <returns>True for success, false for failure.</returns>
public bool SetCLThreshold (string deviceKey, int deviceAddress, int motor, int
   threshold)

/// <summary>
/// This method gets the closed-loop hardware status word from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="hardwareStatus">The hardware status word.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetCLHardwareStatus (string deviceKey, int deviceAddress, int motor, ref uint
   hardwareStatus)

/// <summary>
/// This method sets the closed-loop hardware status word for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <param name="hardwareStatus">The hardware status word.</param>
/// <returns>True for success, false for failure.</returns>
public bool SetCLHardwareStatus (string deviceKey, int deviceAddress, int motor, uint
   hardwareStatus)

/// <summary>
/// This method performs a Move To Home on the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <param name="motor">The motor, or axis, number (1 - 4).</param>
/// <returns>True for success, false for failure.</returns>
public bool MoveToHome (string deviceKey, int deviceAddress, int motor)

/// <summary>
/// This method gets the device address of the specified master controller.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>The device address of the specified master controller.</returns>
public int GetMasterDeviceAddress (string deviceKey)

/// <summary>
/// This method gets the number of slave devices belonging to the specified master
/// controller.
/// </summary>
/// <param name="deviceKey">The device key of the master controller.</param>
/// <returns>The number of slave devices.</returns>
public int GetSlaveCount (string deviceKey)

/// <summary>
/// This method gets the device addresses of the slaves for the specified master
/// controller.
/// </summary>
/// <param name="deviceKey">The device key of the master controller.</param>
/// <returns>The device addresses of the slaves for the specified master
/// controller.</returns>
public int[] GetDeviceAddresses (string deviceKey)

/// <summary>
/// This method gets the number of address conflicts for the specified master controller.
/// </summary>
/// <param name="deviceKey">The device key of the master controller.</param>
/// <returns>The number of address conflicts.</returns>
public int GetAddressConflictCount (string deviceKey)

/// <summary>
/// This method gets the address conflicts for the specified master controller.
/// </summary>
/// <param name="deviceKey">The device key of the master controller.</param>
/// <returns>The address conflicts for the specified master controller.</returns>
public int[] GetAddressConflicts (string deviceKey)

/// <summary>
/// This method determines if the specified device is a master controller.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <returns>True if the specified device is a master controller, otherwise
/// false.</returns>
public bool IsMasterController (string deviceKey, int deviceAddress)

/// <summary>
/// This method determines if the specified device is a slave controller.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <returns>True if the specified device is a slave controller, otherwise
/// false.</returns>
public bool IsSlaveController (string deviceKey, int deviceAddress)
```

```
/// <summary>
/// This method removes the specified slave device from the Device Table.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
public void RemoveSlaveFromDeviceTable (string deviceKey, int deviceAddress)

/// <summary>
/// This method gets a display string for the specified controller.
/// </summary>
/// <param name="deviceKey">The device key</param>
/// <returns>The display string for the specified controller.</returns>
public string GetControllerID (string deviceKey)

/// <summary>
/// This property gets the logging flag.
/// </summary>
public bool IsLogging

/// <summary>
/// This property gets / sets the flag that determines how USB device discovery is
/// performed.
/// True to discover USB devices dynamically using events, otherwise false to discover
/// once during initialization.
/// </summary>
public bool IsUsingUSBEvents

/// <summary>
/// This method sets the product ID to use as a filter for devices to open.
/// </summary>
/// <param name="productID">The product ID.</param>
public void SetUSBProductID (int productID)

/// <summary>
/// This method gets the MAC address of the specified device if it is in the list of
/// discovered devices.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>The MAC address of the discovered device, otherwise an empty
/// string.</returns>
public string GetMACAddress (string deviceKey)

/// <summary>
/// This method sets the rate at which the Ethernet discovery process repeats itself.
/// </summary>
/// <param name="seconds">The number of seconds.</param>
public void SetEthernetDiscoveryRate (int seconds)

/// <summary>
/// This method gets the port type for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>The port type</returns>
public ePortType GetPortType (string deviceKey)

/// <summary>
/// This method gets the underlying port object of the specified port type.
/// </summary>
/// <param name="portType">The port type.</param>
/// <returns>The underlying port object or null for an error.</returns>
public object GetPortObject (ePortType portType)

/// <summary>
/// This method rediscovers the specified device at the application level (the
/// underlying DLLs for specific I/O port types are not affected).
/// </summary>
/// <param name="deviceKey">The device key.</param>
public void RediscoverDevice (string deviceKey)

/// <summary>
/// This method gets the number of discovered devices.
/// </summary>
/// <returns>The number of discovered devices.</returns>
public int GetDeviceCount ()

/// <summary>
/// This method gets the first device key from the list of discovered devices.
/// </summary>
/// <returns>The first device key from the list of discovered devices.</returns>
public string GetFirstDeviceKey ()

/// <summary>
/// This method gets all the device keys from the list of discovered devices.
/// </summary>
/// <returns>All the device keys that have been discovered.</returns>
public string[] GetDeviceKeys ()

/// <summary>
/// This method determines if the passed in device key is valid or not.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>True if the device key is in the list of discovered devices, otherwise
/// false.</returns>
public bool DeviceKeyIsValid (string deviceKey)

/// <summary>
/// This method gets the model and serial number from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>The model and serial number separated by a blank.</returns>
public string GetModelSerial (string deviceKey)

/// <summary>
/// This method gets the model and serial number from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="deviceAddress">The device address.</param>
/// <returns>The model and serial number separated by a blank.</returns>
public string GetModelSerial (string deviceKey, int deviceAddress)

/// <summary>
/// This method sets the model and serial number for the specified device by calling
/// 'IdentifyInstrument'.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>True for success, false for failure.</returns>
public bool SetModelSerial (string deviceKey)

/// <summary>
/// This method gets the display name from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>The display name.</returns>
public string GetDisplayName (string deviceKey, int deviceAddress)

/// <summary>
/// This method sets the display name for the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="displayName">The display name.</param>
/// <returns>True for success, false for failure.</returns>
public bool SetDisplayName (string deviceKey, int deviceAddress, string displayName)

/// <summary>
/// This method identifies the instrument by sending the *IDN? query and parsing the
/// response data.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="model">The model.</param>
/// <param name="serialNum">The serial number.</param>
/// <param name="fwVersion">The firmware version.</param>
/// <param name="fwDate">The firmware date.</param>
```

```
public void IdentifyInstrument (string deviceKey, out string model, out string serialNum,
   out string fwVersion, out string fwDate)

/// <summary>
/// This method parses the information from the instrument identification string.
/// </summary>
/// <param name="identification">The instrument identification string.</param>
/// <param name="model">The model.</param>
/// <param name="serialNum">The serial number.</param>
/// <param name="fwVersion">The firmware version.</param>
/// <param name="fwDate">The firmware date.</param>
/// <returns>True for success, false for failure.</returns>
public bool ParseIDInfo (string identification, out string model, out string serialNum,
   out string fwVersion, out string fwDate)

/// <summary>
/// This method parses the model and serial number from the device key.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="model">The model.</param>
/// <param name="serialNum">The serial number.</param>
/// <returns>True for success, false for failure.</returns>
public bool ParseModelSerial (string deviceKey, ref string model, ref string serialNum)

/// <summary>
/// This method opens the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>True for success, false for failure.</returns>
public bool Open (string deviceKey)

/// <summary>
/// This method closes the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>True for success, false for failure.</returns>
public bool Close (string deviceKey)

/// <summary>
/// This method reads a string response from the device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="value">The string response.</param>
/// <returns>True for success, false for failure.</returns>
public bool Read (string deviceKey, ref string value)

/// <summary>
/// This method reads data from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="buffer">The buffer to hold the response data.</param>
/// <returns>True for success, false for failure.</returns>
public bool Read (string deviceKey, StringBuilder buffer)

/// <summary>
/// This method reads data from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="buffer">The buffer to hold the response data.</param>
/// <param name="nNumBytesRead">The number of bytes read.</param>
/// <returns>True for success, false for failure.</returns>
public bool Read (string deviceKey, byte[] buffer, out int nNumBytesRead)

/// <summary>
/// This method writes data to the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="buffer">The buffer to hold the data.</param>
/// <returns>True for success, false for failure.</returns>
public bool Write (string deviceKey, string buffer)
```

```
/// <summary>
/// This method writes data to the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="buffer">The buffer to hold the data.</param>
/// <param name="dataLength">The length of the data in the buffer.</param>
/// <returns>True for success, false for failure.</returns>
public bool Write (string deviceKey, byte[] buffer, int dataLength)

/// <summary>
/// This method sends the passed in query to the device and returns the string response.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The query string.</param>
/// <param name="value">The string response.</param>
/// <returns>True for success, false for failure.</returns>
public bool Query (string deviceKey, string cmd, ref string value)

/// <summary>
/// This method sends the passed in command string to the specified device
/// and reads the response data.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="command">The command string to send.</param>
/// <param name="buffer">The buffer to hold the response data.</param>
/// <returns>True for success, false for failure.</returns>
public bool Query (string deviceKey, string command, StringBuilder buffer)

/// <summary>
/// This method writes the passed in string to the log file if logging is turned on.
/// </summary>
/// <param name="outputText">The text to output.</param>
public void WriteLog (string outputText)

/// <summary>
/// This method writes the passed in arguments to the log file according
/// to the specified format.
/// </summary>
/// <param name="logging">True if logging, otherwise false.</param>
/// <param name="format">The format specifier.</param>
/// <param name="args">The data values to output.</param>
public void WriteLog (bool logging, string format, params object[] args)

/// <summary>
/// This method writes the passed in arguments to the log file according
/// to the specified format.
/// </summary>
/// <param name="format">The format specifier.</param>
/// <param name="args">The data values to output.</param>
public void WriteLog (string format, params object[] args)

/// <summary>
/// This method reads a string response from the device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="value">The string response.</param>
/// <returns>True for success, false for failure.</returns>
public bool ReadString (string deviceKey, ref string value)

/// <summary>
/// This method sends the passed in command to the device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The command string.</param>
/// <returns>True for success, false for failure.</returns>
public bool WriteString (string deviceKey, string cmd)

/// <summary>
/// This method sends the passed in query to the device and returns the string response.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The query string.</param>
/// <param name="value">The string response.</param>
/// <returns>True for success, false for failure.</returns>
public bool QueryString (string deviceKey, string cmd, ref string value)

/// <summary>
/// This method sends the passed in command to the device along with an integer
/// parameter.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The command string.</param>
/// <param name="value">The integer parameter.</param>
/// <returns>True for success, false for failure.</returns>
public bool WriteInt (string deviceKey, string cmd, int value)

/// <summary>
/// This method sends the passed in query to the device and returns the integer response.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The query string.</param>
/// <param name="value">The integer response.</param>
/// <param name="shouldConvert">True if the response string should be converted to a
/// number, otherwise false.</param>
/// <returns>True for success, false for failure.</returns>
public bool QueryInt (string deviceKey, string cmd, ref int value, bool shouldConvert)

/// <summary>
/// This method sends the passed in command to the device along with an unsigned integer
/// parameter.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The command string.</param>
/// <param name="value">The unsigned integer parameter.</param>
/// <returns>True for success, false for failure.</returns>
public bool WriteUInt (string deviceKey, string cmd, uint value)

/// <summary>
/// This method sends the passed in query to the device and returns the unsigned integer
/// response.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The query string.</param>
/// <param name="value">The unsigned integer response.</param>
/// <param name="shouldConvert">True if the response string should be converted to a
/// number, otherwise false.</param>
/// <returns>True for success, false for failure.</returns>
public bool QueryUInt (string deviceKey, string cmd, ref uint value, bool shouldConvert)

/// <summary>
/// This method sends the passed in command to the device along with a floating point
/// parameter.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The command string.</param>
/// <param name="value">The floating point parameter.</param>
/// <returns>True for success, false for failure.</returns>
public bool WriteFloat (string deviceKey, string cmd, float value)

/// <summary>
/// This method sends the passed in query to the device and returns the floating point
/// response.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The query string.</param>
/// <param name="value">The floating point response.</param>
/// <param name="shouldConvert">True if the response string should be converted to a
/// number, otherwise false.</param>
/// <returns>True for success, false for failure.</returns>
public bool QueryFloat (string deviceKey, string cmd, ref float value, bool
   shouldConvert)
```

```
/// <summary>
/// This method sends the passed in command to the device along with a decimal parameter.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The command string.</param>
/// <param name="value">The decimal parameter.</param>
/// <returns>True for success, false for failure.</returns>
public bool WriteDecimal (string deviceKey, string cmd, decimal value)

/// <summary>
/// This method sends the passed in query to the device and returns the decimal response.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The query string.</param>
/// <param name="value">The decimal response.</param>
/// <param name="shouldConvert">True if the response string should be converted to a
/// number, otherwise false.</param>
/// <returns>True for success, false for failure.</returns>
public bool QueryDecimal (string deviceKey, string cmd, ref decimal value, bool
   shouldConvert)

/// <summary>
/// This method sends the passed in command to the device along with a double-precision
/// parameter.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The command string.</param>
/// <param name="value">The double-precision parameter.</param>
/// <returns>True for success, false for failure.</returns>
public bool WriteDouble (string deviceKey, string cmd, double value)

/// <summary>
/// This method sends the passed in query to the device and returns the double-precision
/// response.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="cmd">The query string.</param>
/// <param name="value">The double-precision response.</param>
/// <param name="shouldConvert">True if the response string should be converted to a
/// number, otherwise false.</param>
/// <returns>True for success, false for failure.</returns>
public bool QueryDouble (string deviceKey, string cmd, ref double value, bool
   shouldConvert)

/// <summary>
/// This method performs clean up for this class.
/// </summary>
public void Shutdown ()
```