Programming Samples User's Manual

For the

New Focus Picomotor Application

Version 2.0.0

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

Table of Contents

1 INTRODUCTION 1

2 C# 1

2.1 MODEL 8742 1
2.1.1 OpenMultipleDevices 1
2.1.2 OpenSingleDevice 2
2.1.3 RelativeMove 2
2.1.4 GetPositionAllSlaves 2
2.2 MODEL 8743 3
2.2.1 OpenMultipleDevices 3
2.2.2 OpenSingleDevice 3
2.2.3 RelativeMove 4
2.2.4 GetPositionAllSlaves 4
2.2.5 EnableClosedLoop 4

3 LABVIEW 5

3.1 MODEL 8742 5
3.1.1 SampleGetIDMultiple.vi 6
3.1.2 SampleGetIDSingle.vi 6
3.1.3 SampleRelativeMove.vi 7
3.1.4 SampleGetPositionAllSlaves 7
3.1.5 Sample8742UI 7
3.2 MODEL 8743 7
3.2.1 SampleGetIDMultiple.vi 8
3.2.2 SampleGetIDSingle.vi 9
3.2.3 SampleRelativeMove.vi 9
3.2.4 SampleGetPositionAllSlaves 9
3.2.5 Sample8743UI 10

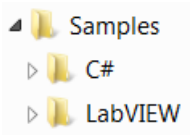
4 CMDLIB 10

4.1 CMDLIB INITIALIZATION 10
4.2 CLEAN UP 11
4.3 LOGGING 11
4.4 RS-485 COMMUNICATION 11
4.5 CMDLIB INTERFACE IN LABVIEW 11
4.6 THE CMDLIB API 13

1 Introduction

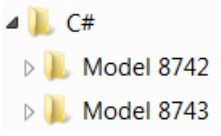
The samples for the Picomotor controller are divided into two folders: C# and LabVIEW. These two folders contain samples that have been developed in the C# programming language and the LabVIEW programming language.

> **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.**



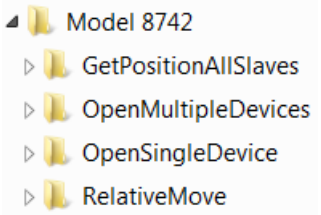
2 C#

The C# folder has a subfolder for each family type supported within the product. 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 Model 8742

The Model 8742 folder has samples for this family of controller that have been developed in the C# programming language. This folder contains a solution file that can be opened in Visual Studio to see the project and the source code for each of the samples.



2.1.1 OpenMultipleDevices

The OpenMultipleDevices sample demonstrates how to communicate with several devices via USB and/or Ethernet in one application. It uses methods in the .NET assembly CmdLib.dll to perform the major steps in this sample. First it calls a CmdLib8742 constructor. This constructor is used to discover multiple instruments. Its function signature is:

```
/// <summary>
/// Constructor.
/// </summary>
/// <param name="logging">True to request logging, otherwise false.</param>
/// <param name="msecDelayForDiscovery">The number of milliseconds to wait for devices to be discovered.</param>
/// <param name="deviceKeys">All the device keys from the list of discovered devices.</param>
public CmdLib8742 (bool logging, int msecDelayForDiscovery, ref string[] deviceKeys)
```

It returns a unique identifier, called a device key, for each device connected via USB or Ethernet. A device key is used to communicate with a particular master controller. Then, for each device key in the list returned by the constructor, 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.

2.1.2 OpenSingleDevice

The OpenSingleDevice sample demonstrates how to communicate with a single device via USB or Ethernet. It uses methods in the .NET assembly CmdLib.dll to perform the major steps in this sample. First it calls a CmdLib8742 constructor. This constructor is used to discover a single device. Its function signature is:

```
/// <summary>
/// Constructor.
/// </summary>
/// <param name="logging">True to request logging, otherwise false.</param>
/// <param name="msecDelayForDiscovery">The number of milliseconds to wait for devices to be discovered.</param>
/// <param name="deviceKey">The first device in the list of open instruments (null = none found).</param>
public CmdLib8742 (bool logging, int msecDelayForDiscovery, ref string deviceKey)
```

It returns a unique identifier, called a device key, for the first device discovered that is connected via USB or Ethernet. A device key is used to communicate with a particular master controller. Then it displays the device key, gets and displays the identification string and shuts down communication.

2.1.3 RelativeMove

The RelativeMove sample demonstrates how to perform motion with a single device via USB or Ethernet. It uses methods in the .NET assembly CmdLib.dll to perform the major steps in this sample. First it calls the same CmdLib8742 constructor that is used in the OpenSingleDevice sample to perform device discovery. Then it 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 and communication is shut down.

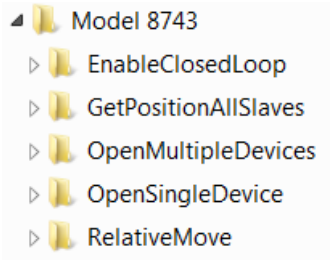
2.1.4 GetPositionAllSlaves

The GetPositionAllSlaves sample demonstrates how to communicate with multiple devices via USB and / or Ethernet and RS-485. It uses methods in the .NET assembly CmdLib.dll to perform the major steps in this sample. When communicating with a slave device, its unique identifier consists of its device address and the device key of its master controller. First it calls the same CmdLib8742 constructor that is used in the OpenMultipleDevices sample to perform device discovery. Then, for each device key in the list returned by the constructor, 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. After all devices have responded, communication is shut down.

2.2 Model 8743

The Model 8742 folder has samples for this family of controller that have been developed in the C# programming language. This folder contains a solution file that can be opened in Visual Studio to see the project and the source code for each of the samples.



2.2.1 OpenMultipleDevices

The OpenMultipleDevices sample demonstrates how to communicate with several devices via USB and/or Ethernet in one application. It uses methods in the .NET assembly CmdLib.dll to perform the major steps in this sample. First it calls a CmdLib8742 constructor. This constructor is used to discover multiple instruments. It returns a unique identifier, called a device key, for each device connected via USB or Ethernet. A device key is used to communicate with a particular master controller. Its function signature is:

```
/// <summary>
/// Constructor.
/// </summary>
/// <param name="logging">True to request logging, otherwise false.</param>
/// <param name="msecDelayForDiscovery">The number of milliseconds to wait for devices to be discovered.</param>
/// <param name="deviceKeys">All the device keys from the list of discovered devices.</param>
public CmdLib8742 (bool logging, int msecDelayForDiscovery, ref string[] deviceKeys)
```

Then, for each device key in the list returned by the constructor, 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.

2.2.2 OpenSingleDevice

The OpenSingleDevice sample demonstrates how to communicate with a single device via USB or Ethernet. It uses methods in the .NET assembly CmdLib.dll to perform the major steps in this sample. First it calls a CmdLib8742 constructor. This constructor is used to discover a single device. It returns a unique identifier, called a device key, for the first device discovered that is connected via USB or Ethernet. A device key is used to communicate with a particular master controller. Its function signature is:

```
/// <summary>
/// Constructor.
/// </summary>
/// <param name="logging">True to request logging, otherwise false.</param>
/// <param name="msecDelayForDiscovery">The number of milliseconds to wait for devices to be discovered.</param>
/// <param name="deviceKey">The first device in the list of open instruments (null = none found).</param>
```

```
public CmdLib8742 (bool logging, int msecDelayForDiscovery, ref string deviceKey)
```

Then it displays the device key, gets and displays the identification string and shuts down communication.

2.2.3 RelativeMove

The RelativeMove sample demonstrates how to perform motion with a single device via USB or Ethernet. It uses methods in the .NET assembly CmdLib.dll to perform the major steps in this sample. First it calls the same CmdLib8742 constructor that is used in the OpenSingleDevice sample to perform device discovery. Then it 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 and communication is shut down.

2.2.4 GetPositionAllSlaves

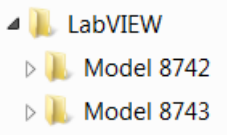
The GetPositionAllSlaves sample demonstrates how to communicate with multiple devices via USB and / or Ethernet and RS-485. It uses methods in the .NET assembly CmdLib.dll to perform the major steps in this sample. When communicating with a slave device, its unique identifier consists of its device address and the device key of its master controller. First it calls the same CmdLib8742 constructor that is used in the OpenMultipleDevices sample to perform device discovery. Then, for each device key in the list returned by the constructor, 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.

2.2.5 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 assembly CmdLib.dll to perform the major steps in this sample. First it calls the same CmdLib8742 constructor that is used in the OpenSingleDevice sample to perform device discovery. Then it 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 and communication is shut down.

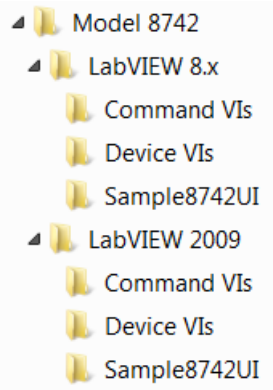
3 LabVIEW

The LabVIEW folder has a subfolder for each family type supported within the product.

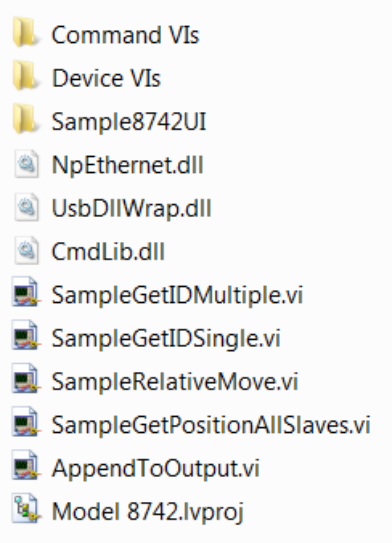


3.1 Model 8742

The Model 8742 folder has samples for this family of controller that have been developed in the LabVIEW programming language. The subfolders (LabVIEW 8.x, and LabVIEW 2009) each contain the same samples written in a different version of LabVIEW.

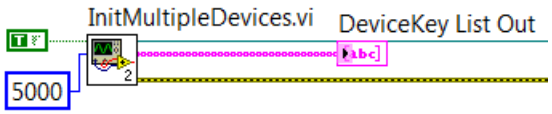


There are samples that demonstrate how to communicate with multiple devices, how to communicate with a single device, how to perform motion, and how to communicate using RS-485. There is also a larger sample in the Sample8743UI folder that shows how to develop an application with a user interface. This sample allows device and motor selection between multiple devices using USB, Ethernet, or RS-485. It controls start / stop motion while properly reporting any errors detected on a slave and its master. The Command VIs folder contains VIs that call a single command in CmdLib.dll. The Device VIs folder contains VIs that call a single generic I/O function (such as open, close, read, write, or query) in CmdLib.dll. 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.



3.1.1 SampleGetIDMultiple.vi

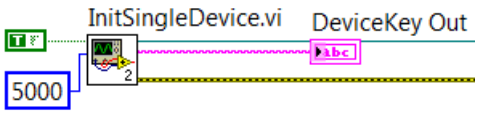
The SampleGetIDMultiple.vi sample demonstrates how to communicate with several devices via USB and/or Ethernet in one application. It uses methods in the .NET assembly CmdLib.dll to perform the major steps in this sample. First it calls InitMultipleDevices.vi. This VI is used to discover multiple instruments.



It returns a unique identifier, called a device key, for each device connected via USB or Ethernet. A device key is used to communicate with a particular master controller. Then, for each device key in the list returned by InitMultipleDevices.vi, it displays the device key, opens the device, gets the device address of the master, gets and displays its identification string, and then closes the device. After all devices have responded, communication is shut down.

3.1.2 SampleGetIDSingle.vi

The SampleGetIDSingle.vi sample demonstrates how to communicate with a single device via USB or Ethernet. It uses methods in the .NET assembly CmdLib.dll to perform the major steps in this sample. First it calls InitSingleDevice.vi. This VI is used to discover a single device.



It returns a unique identifier, called a device key, for the first device discovered that is connected via USB or Ethernet. A device key is used to communicate with a particular master controller. Then it displays the device key, gets and displays the identification string and shuts down communication.

3.1.3 SampleRelativeMove.vi

The SampleRelativeMove.vi sample demonstrates how to perform motion with a single device via USB or Ethernet. It uses methods in the .NET assembly CmdLib.dll to perform the major steps in this sample. First it calls the same InitSingleDevice.vi that is used in the SampleGetIDSingle.vi sample to perform device discovery. Then it gets the device address of the master, sets the position to zero, and performs a relative move. **When the user clicks the LabVIEW Run arrow-button, this sample will move the selected motor the number of steps specified in the "Relative Steps" text box**. Then it 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 and communication is shut down.

3.1.4 SampleGetPositionAllSlaves

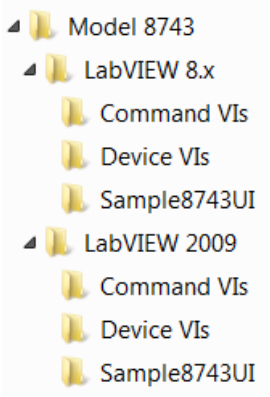
The SampleGetPositionAllSlaves.vi sample demonstrates how to communicate with multiple devices via USB and / or Ethernet and RS-485. It uses methods in the .NET assembly CmdLib.dll to perform the major steps in this sample. When communicating with a slave device, its unique identifier consists of its device address and the device key of its master controller. First it calls the same InitMultipleDevices.vi that is used in the SampleGetIDMultiple.vi sample to perform device discovery. Then, for each device key in the list returned by InitMultipleDevices.vi, 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.

3.1.5 Sample8742UI

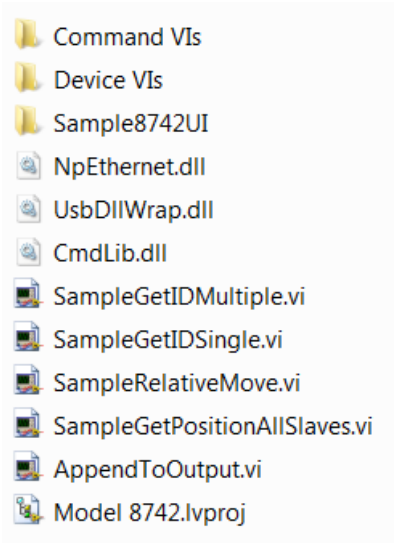
The Sample8742UI.vi sample demonstrates how to develop an application with a user interface that communicates with multiple devices via USB and / or Ethernet and RS-485. It discovers all picomotor controllers connected to the PC and allows any master or slave device to be selected for communication. It dynamically updates the UI to reflect the valid motors that can be selected based upon the selected controller. The user can enter the amount to move and click the Go button to initiate a Relative Move command. The timeout handler is executed every 500 milliseconds and it checks for motion done, displays the current position, and reports any errors detected on a slave and its master. The Stop Motion button will abort motion immediately. When the application is exited the master controller is closed and communication is shut down.

3.2 Model 8743

The Model 8743 folder has samples for this family of controller that have been developed in the LabVIEW programming language. The subfolders (LabVIEW 8.x, and LabVIEW 2009) each contain the same samples written in a different version of LabVIEW.

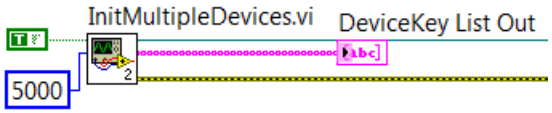


There are samples that demonstrate how to communicate with multiple devices, how to communicate with a single device, how to perform motion, and how to communicate using RS-485. There is also a larger sample in the Sample8743UI folder that shows how to develop an application with a user interface. This sample allows device and motor selection between multiple devices using USB, Ethernet, or RS-485. It controls start / stop motion and closed-loop enable / disable while properly reporting any errors detected on a slave and its master. The Command VIs folder contains VIs that call a single command in CmdLib.dll. The Device VIs folder contains VIs that call a single generic I/O function (such as open, close, read, write, or query) in CmdLib.dll. 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.



3.2.1 SampleGetIDMultiple.vi

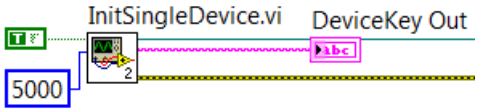
The SampleGetIDMultiple.vi sample demonstrates how to communicate with several devices via USB and/or Ethernet in one application. It uses methods in the .NET assembly CmdLib.dll to perform the major steps in this sample. First it calls InitMultipleDevices.vi. This VI is used to discover multiple instruments.



It returns a unique identifier, called a device key, for each device connected via USB or Ethernet. A device key is used to communicate with a particular master controller. Then, for each device key in the list returned by InitMultipleDevices.vi, it displays the device key, opens the device, gets the device address of the master, gets and displays its identification string, and then closes the device. After all devices have responded, communication is shut down.

3.2.2 SampleGetIDSingle.vi

The SampleGetIDSingle.vi sample demonstrates how to communicate with a single device via USB or Ethernet. It uses methods in the .NET assembly CmdLib.dll to perform the major steps in this sample. First it calls InitSingleDevice.vi. This VI is used to discover a single device.



It returns a unique identifier, called a device key, for the first device discovered that is connected via USB or Ethernet. A device key is used to communicate with a particular master controller. Then it displays the device key, gets and displays the identification string and shuts down communication.

3.2.3 SampleRelativeMove.vi

The SampleRelativeMove.vi sample demonstrates how to perform motion with a single device via USB or Ethernet. It uses methods in the .NET assembly CmdLib.dll to perform the major steps in this sample. First it calls the same InitSingleDevice.vi that is used in the SampleGetIDSingle.vi sample to perform device discovery. Then it gets the device address of the master, sets the position to zero, and performs a relative move. **When the user clicks the LabVIEW Run arrow-button, this sample will move the selected motor the number of steps specified in the "Relative Steps" text box**. Then it 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 and communication is shut down.

3.2.4 SampleGetPositionAllSlaves

The SampleGetPositionAllSlaves.vi sample demonstrates how to communicate with a mixture of open-loop (8742) and closed-loop (8743) devices via USB and / or Ethernet and RS-485. It uses methods in the .NET assembly CmdLib.dll to perform the major steps in this sample. When communicating with a slave device, its unique identifier consists of its device address and the device key of its master controller. First it calls the same InitMultipleDevices.vi that is used in the SampleGetIDMultiple.vi sample to perform device discovery. Then, for each device key in the list returned by InitMultipleDevices.vi, 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.

3.2.5 Sample8743UI

The Sample8743UI.vi sample demonstrates how to develop an application with a user interface that communicates with a mixture of open-loop (8742) and closed-loop (8743) devices via USB and / or Ethernet and RS-485. It discovers all picomotor controllers connected to the PC and allows any master or slave device to be selected for communication. It dynamically updates the UI to reflect the valid motors that can be selected and whether the closed-loop setting can be enabled or disabled based upon the selected controller. The user can enter the amount to move and click the Go button to initiate a Relative Move command. The timeout handler is executed every 500 milliseconds and it checks for motion done, displays the current position, and reports any errors detected on a slave and its master. The Stop Motion button will abort motion immediately. When the application is exited the master controller is closed and communication is shut down.

4 CmdLib

CmdLib.dll is a library of methods (or functions) that can be called by any programming language that supports interfacing with a .NET library. Using CmdLib.dll reduces the amount of work that it takes to communicate with a picomotor controller. This library has methods to discover all picomotors connected to a PC, to return information about discovered controllers, to execute picomotor commands, and to perform general device I/O (such as open, close, read, write, and query). This library has a public method for most of the commands described in the user's manual for the instrument. If a particular command is not implemented in CmdLib.dll then a Read, Write, or Query method in CmdLib.dll can be used to execute any of the commands described in the user's manual.

In general, there are only two steps required in order to communicate with a picomotor controller using CmdLib.dll. The first step is to perform device initialization by calling the constructor, and the second step is to send a command to the device. The third step is not required to communicate, but the Shutdown method should be called before the program exits.

4.1 CmdLib Initialization

A CmdLib8742 constructor initializes communication with one or more picomotor controllers. It performs device discovery on all instruments connected via USB or Ethernet. The master controllers are connected by USB or Ethernet and the slave controllers are daisy chained to a master by RS-485. Each master controller is uniquely identified by a "device key" that is used by CmdLib.dll to communicate with that particular device. Each slave controller is uniquely identified by its device address and the device key of its master. One constructor performs device discovery and returns the first device key in the list. The other constructor performs device discovery and returns all device keys in the list. The number of milliseconds to delay for discovery allows the discovery process to complete before attempting to open any device for communication. USB discovery is very quick and does not usually need a delay, but Ethernet

discovery needs around 2000 - 5000 milliseconds. Logging can be turned on or off with the Boolean value passed into the constructor. Logging 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.

4.2 Clean Up

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

4.3 Logging

Logging can be turned on by passing a Boolean true value into the CmdLib8742 constructor. 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.

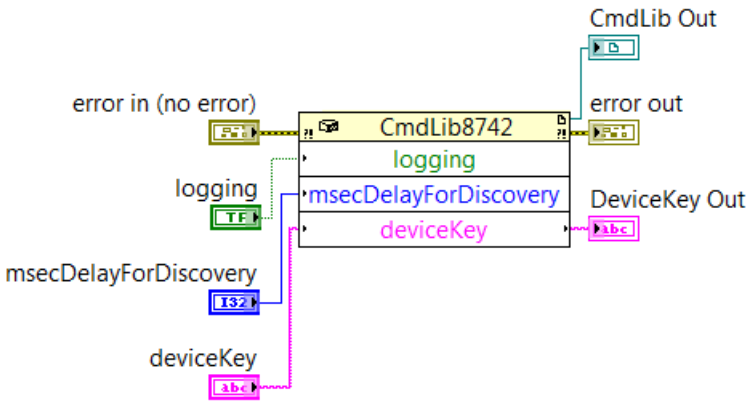
4.4 RS-485 Communication

The only additional requirement for RS-485 communication is to specify the device address of the controller. There is nothing extra to be done for initialization beyond what was specified in section 4.1, however the method '`GetDeviceAddresses`' can be used to retrieve a list of the device addresses used by the slaves of the specified master controller (see the GetPositionAllSlaves sample or the more complex GetDiscoveredDevices.vi in Sample8743UI). CmdLib.dll has several methods with two signatures (one with a device address parameter, and one without it). The signature that does not specify a device address will talk to the master controller. The signature that does specify a device address will talk to the controller using that address. If a Read, Write, or Query method is used then the command string that is sent to the device must include the device address followed by a greater than sign and then the command string (e.g. 2>*IDN?).

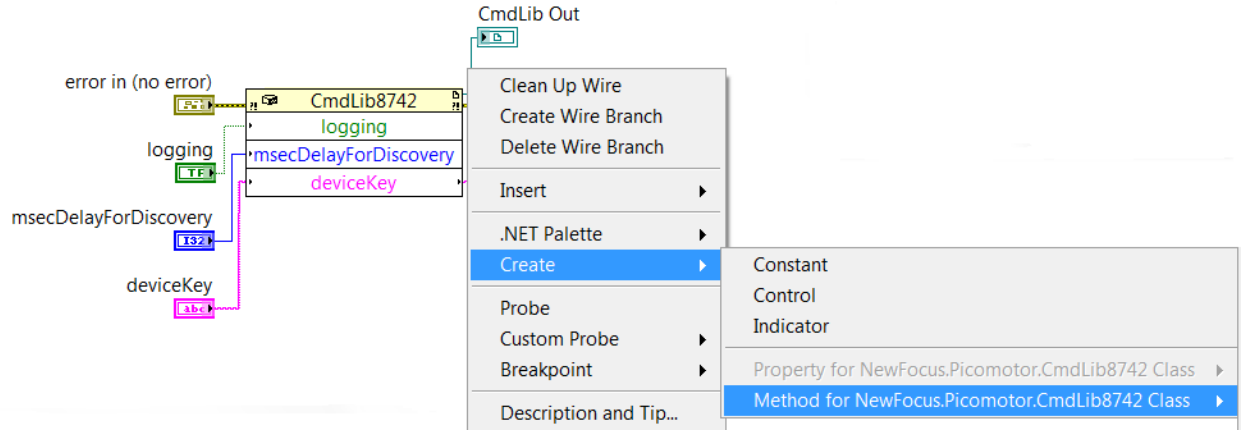
4.5 CmdLib Interface in LabVIEW

Since LabVIEW supports calling methods in a .NET library, any of the constructors and methods in CmdLib.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 CmdLib.dll. 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 CmdLib.dll may exist to perform this command. If neither exist, then a Read, Write, or Query method in CmdLib.dll 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).

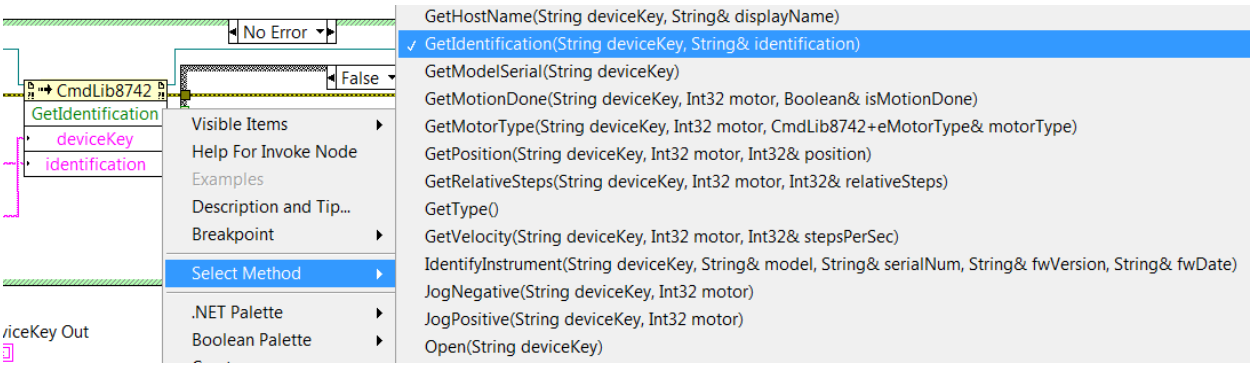
As stated in section 4 above, there are two basic steps necessary to send a command or query to the instrument. The first step is to initialize device communication. This can be done by calling InitSingleDevice.vi or InitMultipleDevices.vi (from the Device VIs folder) or by directly calling the LabVIEW constructor block that is wrapped inside one of these two VIs. The CmdLib8742 constructor block that is called by InitSingleDevice.vi is shown below. The CmdLib8742 constructors are described in section 4.1.



The second 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 CmdLib.dll. To call a method in CmdLib.dll, simply right-click the upper right hand corner of any CmdLib8742 block (on the CmdLib Out wire) to display a context menu and select "Create". Then select "Method for NewFocus.Picomotor.CmdLib8742 Class", and select the name of the method to be called from the list in the context menu.



If a LabVIEW method block 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.



4.6 The CmdLib API

This section describes the public interface for CmdLib.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 ()

/// <summary>
/// Constructor.
/// </summary>
/// <param name="logging">True to request logging, otherwise false.</param>
/// <param name="msecDelayForDiscovery">The number of milliseconds to wait for devices to be discovered.</param>
/// <param name="deviceKey">The first device in the list of open instruments (null = none found).</param>
public CmdLib8742 (bool logging, int msecDelayForDiscovery, ref string deviceKey)

/// <summary>
/// Constructor.
/// </summary>
/// <param name="logging">True to request logging, otherwise false.</param>
/// <param name="msecDelayForDiscovery">The number of milliseconds to wait for devices to be discovered.</param>
/// <param name="deviceKeys">All the device keys from the list of discovered devices.</param>
public CmdLib8742 (bool logging, int msecDelayForDiscovery, ref string[] deviceKeys)

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

/// <summary>
/// This method sets the retry timeout value for the devices during discovery.
/// </summary>
/// <param name="seconds">The number of seconds.</param>
public void SetDiscoveryRetryTimeout (int seconds)

/// <summary>
/// This method discovers the devices that are available for communication.
/// </summary>
public void DiscoverDevices ()
```

```
/// <summary>
/// This method rediscovers the specified device at the application level (the
/// underlying DLLs for USB and Ethernet 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 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 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 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 returns a fixed unique identifier for the specified device.  If the device key
/// is fixed, then that is returned.  In the case where the device key is an IP address, the
/// MAC address is returned.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <returns>The fixed unique identifier for the specified device.</returns>
```

```
        public string GetFixedID (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 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 model and serial number from the instrument identification string.
        /// </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 performs the required steps to cleanly stop communication.
/// </summary>
public void Shutdown ()
```