Programming Samples User's Manual

For the

New Focus Chopper Application

Version 1.0.0

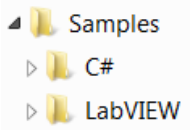
Prepared by:
New Focus
3635 Peterson Way
Santa Clara, CA 95054

Table of Contents

1 Introduction

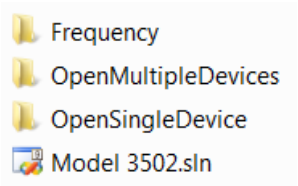
The samples for the Chopper controller are divided into two folders: C# and LabVIEW. These two folders contain samples that have been developed in the C# programming language and the LabVIEW programming language.

> **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 sample that has 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. 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 in one application. It uses methods in the .NET assembly CmdLib3502.dll to perform the major steps in this sample. First it calls the CmdLib3502 constructor to initialize the command library. Its function signature is:

```
/// <summary>
/// Constructor.
/// </summary>
/// <param name="isLogging">True to turn on logging, otherwise false.</param>
public CmdLib3502 (bool isLogging)
```

Then it calls DiscoverDevices to discover multiple instruments, and GetDeviceKeys to get a list of the discovered devices. This list contains a unique identifier, called a device key, for each device connected via USB. A device key is used to communicate with a particular instrument. Then, for each device key in the list, it displays the device key, and then gets and displays its identification string. After all devices have responded, communication is shut down.

2.2 OpenSingleDevice

The OpenSingleDevice sample demonstrates how to communicate with a single device via USB. It uses methods in the .NET assembly CmdLib3502.dll to perform the major steps in this sample.

First it calls the CmdLib3502 constructor to initialize the command library. Its function signature is:

```
/// <summary>
/// Constructor.
/// </summary>
/// <param name="isLogging">True to turn on logging, otherwise false.</param>
public CmdLib3502 (bool isLogging)
```

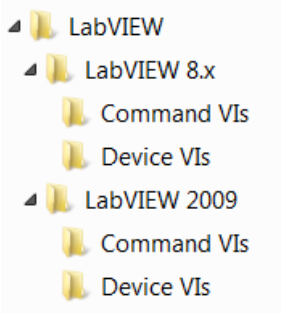
Then it calls DiscoverDevices to discover multiple instruments, and GetFirstDeviceKey to get the first device key from the list of the discovered devices. This list contains a unique identifier, called a device key, for each device connected via USB. A device key is used to communicate with a particular instrument. Then it displays the device key, gets and displays its identification string and shuts down communication.

2.3 Frequency

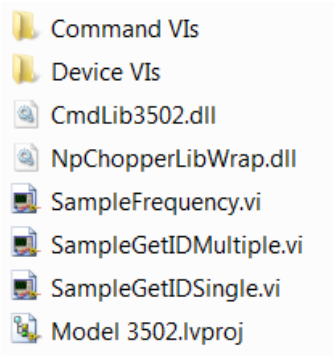
The Frequency sample demonstrates how to get and set the frequency for a single device via USB. It uses methods in the .NET assembly CmdLib3502.dll to perform the major steps in this sample. First it calls the CmdLib3502 constructor to initialize the command library. Then it calls DiscoverDevices to discover multiple instruments, and GetFirstDeviceKey to get the first device key from the list of the discovered devices. Then it calls GetSynthFrequency to get the synthesizer frequency and display it before it is set by the sample. Then it calls SetSynthFrequency to set the synthesizer frequency, and GetSynthFrequency to verify that it has been changed. Then it calls GetFrequency four times, each with a different index (1 – 4), to get a frequency and display it. Finally, it shuts down communication.

3 LabVIEW

The LabVIEW folder has samples 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.

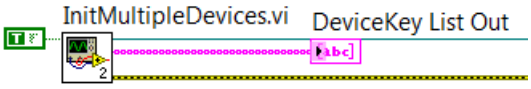


There are samples that demonstrate how to communicate with multiple devices, how to communicate with a single device and how to get / set the frequency. The Command VIs folder contains VIs that call a single command in CmdLib3502.dll. The Device VIs folder contains VIs that call a single generic I/O function (such as read, write, or query) in CmdLib3502.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 in one application. It uses methods in the .NET assembly CmdLib3502.dll to perform the major steps in this sample. First it calls InitMultipleDevices.vi. This VI is used to discover multiple instruments.

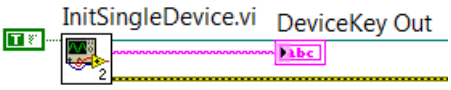


It returns a unique identifier, called a device key, for each device connected via USB. A device key is used to communicate with a particular instrument. Then, for each device key in the list

returned by InitMultipleDevices.vi, it displays the device key, and then gets and displays its identification string. 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. It uses methods in the .NET assembly CmdLib3502.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. A device key is used to communicate with a particular instrument. Then it displays the device key, gets and displays the identification string and shuts down communication.

3.1.3 SampleFrequency.vi

The SampleFrequency.vi sample demonstrates how to get and set the frequency for a single device via USB. It uses methods in the .NET assembly CmdLib3502.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 calls GetSynthFrequency.vi to get the synthesizer frequency and display it before it is set by the sample. Then it calls SetSynthFrequency.vi to set the synthesizer frequency, and GetSynthFrequency.vi to verify that it has been changed. Then it calls GetFrequency four times, each with a different index (1 – 4), to get a frequency and display it. Finally, it shuts down communication.

4 CmdLib3502

CmdLib3502.dll is a library of methods (or functions) that can be called by any programming language that supports interfacing with a .NET library. Using CmdLib3502.dll reduces the amount of work that it takes to communicate with a chopper controller. This library has methods to discover all choppers connected to a PC, to return information about discovered controllers, to execute chopper commands, and to perform general device I/O (such as 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 CmdLib3502.dll then a Read, Write, or Query method in CmdLib3502.dll can be used to execute any of the commands described in the user's manual.

In general, there are only a few steps required in order to communicate with a chopper controller using CmdLib3502.dll. The first step is to perform device initialization (by calling the constructor, discovering devices, and getting the device keys). 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 CmdLib3502 Initialization

The CmdLib3502 constructor initializes communication with one or more chopper controllers. The constructor has one parameter that allows the user to turn logging on or off. 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. The DeviceDiscovery method performs device discovery on all instruments that are powered on and attached to the PC by USB cable. Each instrument is uniquely identified by a "device key" that is used by CmdLib3502.dll to communicate with that particular device. The GetDeviceKeys method can be used to retrieve the device key for all discovered devices, or GetFirstDeviceKey can be called when only one instrument is connected to the PC.

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.

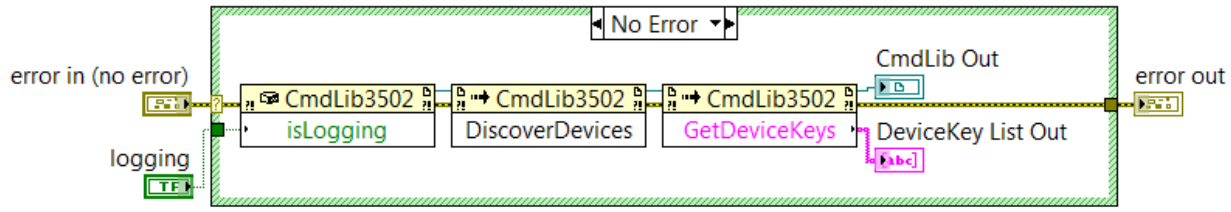
4.3 Logging

Logging can be turned on by passing a Boolean true value into the CmdLib3502 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.

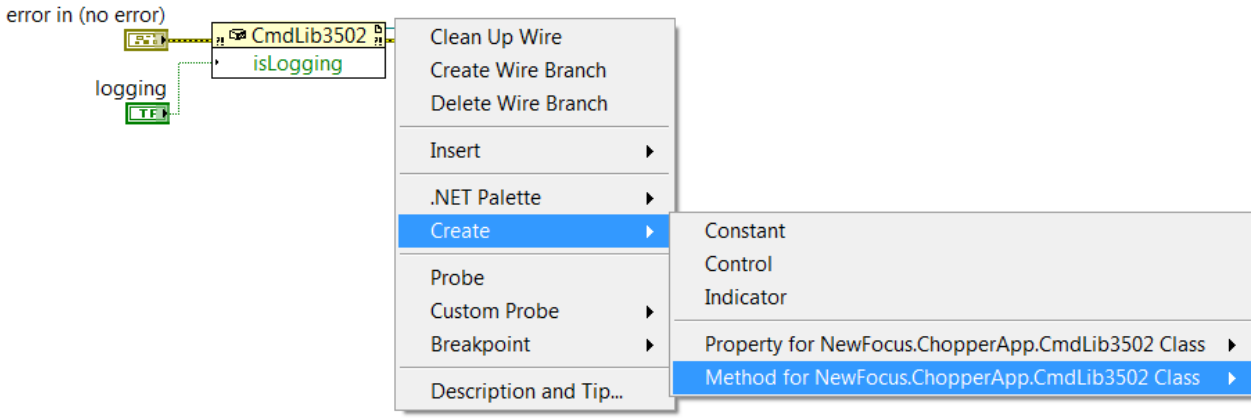
4.4 CmdLib3502 Interface in LabVIEW

Since LabVIEW supports calling methods in a .NET library, any of the constructors and methods in CmdLib3502.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 CmdLib3502.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 CmdLib3502.dll may exist to perform this command. If neither exist, then a Read, Write, or Query method in CmdLib3502.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 read, write, and query).

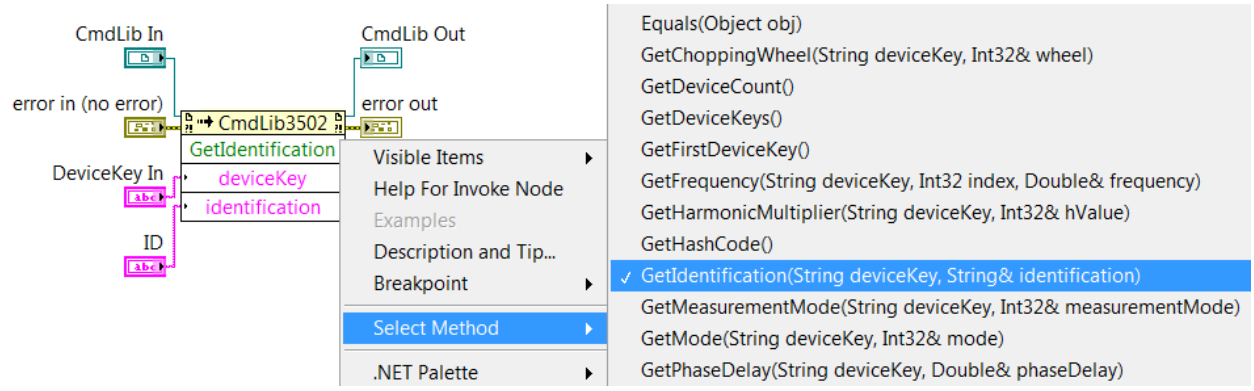
As stated in section 4 above, there are a few basic steps necessary to send a command or query to the instrument. The first step is to initialize device communication. This can be done by calling InitSingleDevice.vi or InitMultipleDevices.vi (from the Device VIs folder) or by directly calling the library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



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 CmdLib3502.dll. To call a method in CmdLib3502.dll, simply right-click the upper right hand corner of any CmdLib3502 block (on the CmdLib Out wire) to display a context menu and select "Create". Then select "Method for NewFocus.ChopperApp.CmdLib3502 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.



4.5 The CmdLib3502 API

This section describes the public interface for CmdLib3502.dll. The following constructors and methods are available to any programming language that supports calling a function within a .NET library.

```
/// <summary>
/// Constructor.
/// </summary>
/// <param name="isLogging">True to turn on logging, otherwise false.</param>
public CmdLib3502 (bool isLogging)

/// <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 frequency from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="index">The command index (1 - 4).</param>
/// <param name="frequency">The frequency.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetFrequency (string deviceKey, int index, ref double frequency)

/// <summary>
/// This method gets the harmonic multiplier.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="hValue">The H value (1 - 15).</param>
/// <returns>True for success, false for failure.</returns>
public bool GetHarmonicMultiplier (string deviceKey, ref int hValue)

/// <summary>
/// This method sets the harmonic multiplier.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="hValue">The H value (1 - 15).</param>
/// <returns>True for success, false for failure.</returns>
public bool SetHarmonicMultiplier (string deviceKey, int hValue)

/// <summary>
/// This method sends a key pressed action to the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="index">The command index (0 - 8).</param>
/// <returns>True for success, false for failure.</returns>
public bool SendKey (string deviceKey, int index)

/// <summary>
/// This method stores / recalls an instrument configuration to / from memory of the
/// specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="index">The command index (0 - 1).</param>
/// <returns>True for success, false for failure.</returns>
public bool MemStoreRecall (string deviceKey, int index)

/// <summary>
/// This method gets the mode of operation.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="mode">The mode (0 - 2).</param>
/// <returns>True for success, false for failure.</returns>
public bool GetMode (string deviceKey, ref int mode)
```

```
/// <summary>
/// This method sets the mode of operation.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="mode">The mode (0 - 2).</param>
/// <returns>True for success, false for failure.</returns>
public bool SetMode (string deviceKey, int mode)

/// <summary>
/// This method gets the measurement mode being used.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="measurementMode">The measurement mode (0 - 3).</param>
/// <returns>True for success, false for failure.</returns>
public bool GetMeasurementMode (string deviceKey, ref int measurementMode)

/// <summary>
/// This method sets the measurement mode to use.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="measurementMode">The measurement mode (0 - 3).</param>
/// <returns>True for success, false for failure.</returns>
public bool SetMeasurementMode (string deviceKey, int measurementMode)

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

/// <summary>
/// This method sets the synthesizer frequency from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="frequency">The frequency.</param>
/// <returns>True for success, false for failure.</returns>
public bool SetSynthFrequency (string deviceKey, double frequency)

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

/// <summary>
/// This method sets the phase delay from the specified device.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="phaseDelay">The phase delay.</param>
/// <returns>True for success, false for failure.</returns>
public bool SetPhaseDelay (string deviceKey, double phaseDelay)

/// <summary>
/// This method gets the selected instrument set-up number
/// that was used to recall a formerly stored chopper set-up.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="index">The command index (0 - 9).</param>
/// <returns>True for success, false for failure.</returns>
public bool GetRecall (string deviceKey, ref int index)

/// <summary>
/// This method selects a formerly stored instrument set-up.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="index">The command index (0 - 9).</param>
```

```
/// <returns>True for success, false for failure.</returns>
public bool SetRecall (string deviceKey, int index)

/// <summary>
/// This method selects a formerly stored instrument set-up for recall
/// and performs the recall operation.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="index">The command index (0 - 9), where 0 is restore factory
/// defaults.</param>
/// <returns>True for success, false for failure.</returns>
public bool Recall (string deviceKey, int index)

/// <summary>
/// This method selects the parameter to be modified by the
/// arrow keys on the front panel of the device by setting the
/// state of the Set button.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="index">The command index (0 - 5).</param>
/// <returns>True for success, false for failure.</returns>
public bool SetSetButton (string deviceKey, int index)

/// <summary>
/// This method gets the 8-bit status register.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="statusReg">The 8-bit status register.</param>
/// <returns>True for success, false for failure.</returns>
public bool GetStatusReg (string deviceKey, ref uint statusReg)

/// <summary>
/// This method gets the selected instrument set-up number
/// that was used to store the chopper set-up.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="index">The command index (1 - 9).</param>
/// <returns>True for success, false for failure.</returns>
public bool GetStore (string deviceKey, ref int index)

/// <summary>
/// This method selects the instrument set-up number in which to store
/// the current chopper set-up.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="index">The command index (1 - 9).</param>
/// <returns>True for success, false for failure.</returns>
public bool SetStore (string deviceKey, int index)

/// <summary>
/// This method selects the instrument set-up number in which to store
/// the current chopper set-up and performs the storage operation.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="index">The command index (1 - 9).</param>
/// <returns>True for success, false for failure.</returns>
public bool Store (string deviceKey, int index)

/// <summary>
/// This method gets the subharmonic divide ratio.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="sValue">The S value (1 - 15).</param>
/// <returns>True for success, false for failure.</returns>
public bool GetSubharmonicDivideRatio (string deviceKey, ref int sValue)

/// <summary>
/// This method sets the subharmonic divide ratio.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="sValue">The S value (1 - 15).</param>
```

```
/// <returns>True for success, false for failure.</returns>
public bool SetSubharmonicDivideRatio (string deviceKey, int sValue)

/// <summary>
/// This method gets the type of sync being used.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="sync">The type of sync being used (0 - 3).</param>
/// <returns>True for success, false for failure.</returns>
public bool GetSync (string deviceKey, ref int sync)

/// <summary>
/// This method sets the type of sync being used.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="sync">The type of sync being used (0 - 3).</param>
/// <returns>True for success, false for failure.</returns>
public bool SetSync (string deviceKey, int sync)

/// <summary>
/// This method gets the type of chopping wheel being used.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="wheel">The type of chopping wheel being used (0 - 3).</param>
/// <returns>True for success, false for failure.</returns>
public bool GetChoppingWheel (string deviceKey, ref int wheel)

/// <summary>
/// This method sets the type of chopping wheel being used.
/// </summary>
/// <param name="deviceKey">The device key.</param>
/// <param name="wheel">The type of chopping wheel being used (0 - 3).</param>
/// <returns>True for success, false for failure.</returns>
public bool SetChoppingWheel (string deviceKey, int wheel)

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

/// <summary>
/// This method gets the underlying port object of the USB port.
/// </summary>
/// <returns>The underlying port object or null for an error.</returns>
public object GetPortObject ()

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

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