LC.400 API Developers Manual

(Document Version 1.0, 18 OCT 2019)

3030 Laura Lane, STE 100
Middleton, WI 53562, USA
Phone: 608-824-1770
Fax: 608-824-1774
https://www.npoint.com
support@npoint.com

1 Introduction

The nPoint LC.400 API exports C language functions from a DLL. The API facilitates programming controller internal DSP functionality including a variety of positioning functions, reading the sensor monitor, recording values to internal buffers, and more. The API C language functions can be called from most development environments including Microsoft Visual Studio .NET C++/C#/VB, Python, and Matlab. Provided programming examples are described in Section 3.

1.1 Requirements

1. Windows 7/8/10 PC.
2. USB connection to LC.400, LC.402, or LC.403 controller.
3. FTDI D2XX USB Drivers installed. The latest WHQL Certified setup executable can be downloaded in the "Comments" column of the drivers table here: https://www.ftdichip.com/Drivers/D2XX.htm
4. The standard set of files is 64 bit for x64 platform development. Alternatively a 32 bit set of files is provided for x86 development. The x86 files will work on x64 platforms, but the native x64 files will execute faster if 32 bit compatibility is not a concern.

1.2 API Function Return Codes

Each function call returns a 32 bit signed integer with a value of zero if the function executed successfully or one of the following error codes:

0 – Successful
1 – FTDI USB invalid handle
2 – FTDI USB device not found
3 – FTDI USB device not opened
4 – FTDI USB I/O error
5 – FTDI USB insufficient resources
6 – FTDI USB invalid parameter
16 – FTDI USB invalid arguments
17 – FTDI USB not supported
18 – FTDI USB other
19 – FTDI USB device list not ready
20 – API exception
21 – API parameter out of range
23 – Query return message is incorrect format
24 – More bytes than expected in the receive buffer
25 – Timeout
26 – Connection ID out of bounds
27 – Maximum simultaneous connections reached

2 API Functions

2.1 Connections and Device Information

2.1.1 buildAvailUSB_devList

Summary
Builds an internal list of nPoint LC.400, LC.402, and LC.403 controllers that are available to connect to.

Definition
```
LONG buildOpenUSB_devList(PLONG numDevices)
```

Parameters

numDevices	Pointer to a variable of type LONG which receives the number of available devices in the list.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
This function must be called prior to the getAvailUSB_devInfo and connectUSB functions. If an application has an open USB connection to a controller, it will not be part of this list since only one application can connect to the controller at a time.

2.1.2 getAvailUSB_devInfo

Summary
Gets the serial number and device description for one of the devices in the internal list generated by the buildAvailUSB_devList function.

Definition
```
LONG getAvailUSB_devInfo(LONG deviceIndex, PCHAR serialNumber, PCHAR deviceDescription)
```

Parameters

deviceIndex	Specifies the device to get information for based on the index of the list generated by the buildAvailUSB_devList function. The index value for the first device in the list is zero.
serialNumber	Pointer to a buffer to store the serial number of the USB device as a null-terminated string. The character array should have 16 elements.
deviceDescription	Pointer to a buffer to store the description of the USB device as a null-terminated string, for example “nPoint LC.400 Controller”. The character array should have 64 elements.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
It is required that buildAvailUSB_devList is called prior to calling this function. Typically the buildAvailUSB_devList function is called when an application starts or the user wants to refresh the UI list of available nPoint USB devices.

Then using the numDevices parameter returned by the buildAvailUSB_devList function, getAvailUSB_devInfo is called in a loop to iterate though the list retrieving serial numbers and descriptions.

2.1.3 connectUSB

Summary
Opens a USB connection to an LC.400, LC.402, or LC.403 controller.

Definition
```
LONG connectUSB(LONG USBListIndex, PLONG connectionID)
```

Parameters

USBListIndex	Specifies the controller to connect to based on the index of the list generated by the buildAvailUSB_devlist function.
connectionID	Pointer to a variable of type LONG which receives the ID number that is used to reference this connection for future function calls.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
The API can have multiple simultaneous controller connections open, and each has a unique connectionID so that functions such as getSensorMonitor can identify which connection to use.

2.1.4 disconnect

Summary
Closes a connection to an LC.400, LC.402, or LC.403 controller.

Definition
```
LONG disconnect(LONG connectionID)
```

Parameters

connectionID	Specifies the ID number of the connection to be closed.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
connectionID was originally assigned and returned when the connection was opened by the connect function.

2.1.5 getNumChannels

Summary
Queries the number of controller channels.

Definition
```
LONG getNumChannels(LONG connectionID, PLONG numChannels)
```

Parameters

connectionID	Specifies the ID number of the connection.
numChannels	Pointer to a variable of type LONG which receives the number of physical controller channels.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
An LC.400 returns a value of 1, LC.402 returns a value of 2, and LC.403 returns a value of 3.

2.1.6 getRange

Summary
Queries the range and distance units enumeration value for a single controller channel.

Definition
```
LONG getRange(LONG connectionID, LONG channel, PFLOAT range, PLONG distanceUnits)
```

Parameters

connectionID	Specifies the ID number of the connection.
channel	Specifies the controller channel number. Valid value range is 1 to 3.
range	Pointer to a variable of type FLOAT which receives the travel range of the stage axis connected to the controller channel.
distanceUnits	Pointer to a variable of type LONG which receives the distance units enumeration value of the stage axis connected to the controller channel. 0 = “microns” 1 = “µradians” 2 = “millimeters” 5 = “milliradians”

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
For example an nPFocus 400 linear stage would return values of 400 and 0 for “400 microns”, which would have a position command range of -200 µm to +200 µm.

2.2 Static Position and Individual Sensor Reading

2.2.1 setDigPosition

Summary
Write a new Digital Position Command for a single controller channel.

Definition
```
LONG setDigPosition(LONG connectionID, LONG channel, FLOAT position)
```

Parameters

connectionID	Specifies the ID number of the connection.
channel	Specifies the controller channel number. Valid value range is 1 to 3.
position	Position command for the stage axis in distance units. The center position is zero, the maximum position is half the travel range, the minimum position is (maximum * -1).

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
Writing a new Digital Position command changes the command set point instantaneously. In closed loop (Servo On) the control loop will move the stage to the new position as quickly as possible based on the control loop bandwidth. In open loop (Servo Off) the stage will move at the maximum slew rate of the controller driver circuit. For large changes in position this may cause overshoot or excite mechanical resonances in the positioning system. It is recommended to use a trajectory move with a velocity limit for large closed loop moves, and for open loop moves in general to avoid unnecessary harshness.

2.2.2 getDigPosition

Summary
Queries the current Digital Position Command value for a single controller channel.

Definition
```
LONG getDigPosition(LONG connectionID, LONG channel, PFLOAT position)
```

Parameters

connectionID	Specifies the ID number of the connection.
channel	Specifies the controller channel number. Valid value range is 1 to 3.
position	Pointer to a variable of type FLOAT which receives the Digital Position Command.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
The Digital Position Command value can be set directly by the setDigPosition function or changed by the Trajectory, Spiral Scan, and Raster Scan internal controller functions.

2.2.3 getSensorMonitor

Summary
Queries the current Sensor Monitor value for a single controller channel.

Definition
```
LONG getSensorMonitor(LONG connectionID, LONG channel, PFLOAT sensorMonitor)
```

Parameters

connectionID	Specifies the ID number of the connection.
channel	Specifies the controller channel number. Valid value range is 1 to 3.
sensorMonitor	Pointer to a variable of type FLOAT which receives the Sensor Monitor value.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
The Sensor Monitor value is in distance units such as microns.

2.3 Trajectory Move

2.3.1 setTrajEnable

Summary
Write the internal trajectory function enable state for a single controller channel.

Definition
```
LONG setTrajEnable(LONG connectionID, LONG channel, LONG enableState)
```

Parameters

connectionID	Specifies the ID number of the connection.
channel	Specifies the controller channel number. Valid value range is 1 to 3.
enableState	Enable state value of 1 = enabled, 0 = disabled.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

2.3.2 getTrajEnable

Summary
Queries the internal trajectory function enable state for a single controller channel.

Definition
```
LONG getTrajEnable(LONG connectionID, LONG channel, PLONG enableState)
```

Parameters

connectionID	Specifies the ID number of the connection.
channel	Specifies the controller channel number. Valid value range is 1 to 3.
enableState	Pointer to a variable of type LONG which receives the trajectory enable state value. Enable state value of 1 = enabled, 0 = disabled.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

2.3.3 singleTrajMove

Summary
Moves one or more stage axes using the controller internal trajectory function. This function moves the stage relative to the current Digital Position Command for each axis.

Definition
```
LONG singleTrajMove(LONG connectionID, FLOAT ch1Pos, FLOAT ch2Pos, FLOAT ch3Pos, FLOAT vLimit, FLOAT
aLimit, FLOAT jLimit, FLOAT dwellTime)
```

Parameters

connectionID	Specifies the ID number of the connection.
ch1Pos	Specifies the Ch1 Digital Position Command to move to in distance units, e.g. microns. Use a value of zero if Ch1 has the trajectory function disabled.
ch2Pos	Specifies the Ch2 Digital Position Command to move to. If the controller does not have a second channel, or the trajectory function is disabled for this channel use a value of zero.
ch3Pos	Specifies the Ch3 Digital Position Command to move to. If the controller does not have a second channel, or the trajectory function is disabled for this channel use a value of zero.
vLimit	Specifies the Velocity Limit to use for the move in distance units per millisecond, e.g. µm/ms.
aLimit	Specifies the Acceleration Limit to use for the move in distance units per millisecond squared, e.g. µm/ms².
jLimit	Specifies the Jerk Limit to use for the move in distance units per millisecond cubed, e.g. µm/ms³
dwellTime	Specifies the Dwell Time in seconds for the trajectory function at the destination position. Typically this value is zero for single moves, but it can be useful with the trajectory TTL output function.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
A trajectory move with Velocity Limit is recommended to soften open loop moves or large closed loop moves.

2.3.4 getTrajStatus

Summary
Queries status of the trajectory move to determine if the move is complete.

Definition
```
LONG getTrajStatus(LONG connectionID, PLONG status)
```

Parameters

connectionID	Specifies the ID number of the connection.
status	Pointer to a variable of type LONG which receives the trajectory status value. Status value of 1 = move in progress, 0 = move complete or stopped.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

2.3.5 stopTraj

Summary
Writes the command to stop the controller internal trajectory function.

Definition
```
LONG stopTraj(LONG connectionID)
```

Parameters

connectionID	Specifies the ID number of the connection.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
When the trajectory in progress is stopped with this function, the stage stops at the current position mid-move.

2.4 Arbitrary Waveforms

2.4.1 loadWaveform

Summary
Writes one period of arbitrary waveform data points to a channel. If multiple iterations are used, the controller internal function will loop back to the first point after outputting the last point to behave as a periodic waveform.

Definition
```
LONG loadWaveform(LONG connectionID, LONG channel, PFLOAT waveformData[],LONG waveformDataLength, LONG loopCyclesPerPoint, LONG iterations)
```

Parameters

connectionID	Specifies the ID number of the connection.
channel	Specifies the controller channel number. Valid value range is 1 to 3.
waveformData[]	Pointer to an array of type FLOAT that holds the waveform data points to be written to the controller. Each point is a position in distance units.
waveformDataLength	Specifies the length of the waveformData array. The waveformData array should not be larger than 83,333 points.
loopCyclesPerPoint	Specifies the number of 24µs control loop cycles to wait before the wavetable function outputs the next point. For example a value of 0 will have a 2 second period for a waveformData[] buffer of 83,333 points, and a value of 1 will have a 4 second period.
iterations	Specifies the number of waveform periods to execute prior to automatically setting Waveform Active to zero at the last point in waveformData[].

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
If infinite iterations mode is specified for a channel when calling the simWaveformStart function, the iterations value is ignored and the waveform will run until stopped via software command or TTL trigger. If using a finite number of iterations, when the iterations are complete the last point in waveformData[] will be summed with the Digital Position Command until Waveform Enable is set to zero. The function stopCh1Ch2Ch3Waveforms sets both Waveform Enable and Waveform active to zero for Ch1 Ch2 and Ch3.

2.4.2 simWaveformStart

Summary
Simultaneously starts the arbitrary waveform function for the specified channels so that the phase is synchronized.

Definition
```
LONG simWaveformStart(LONG connectionID, LONG ch1Start, LONG ch2Start, LONG ch3Start)
```

Parameters

connectionID	Specifies the ID number of the connection.
ch1Start	Specifies the arbitrary waveform start option for Ch1: 0 = do not start, 1 = start with infinite iterations, 2 = start with specified finite number of iterations.
ch2Start	Specifies the arbitrary waveform start option for Ch2: 0 = do not start, 1 = start with infinite iterations, 2 = start with specified finite number of iterations.
ch3Start	Specifies the arbitrary waveform start option for Ch3: 0 = do not start, 1 = start with infinite iterations, 2 = start with specified finite number of iterations.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
This function is used for single channel controllers and two channel controllers by using a start parameter value of zero for channels that don't exist. This function sets Waveform Enable to a value of 1 for channels with infinite iterations, Waveform Enable to a value of 2 for channels with finite iterations, and Waveform Active to a value of 1 for any channel that is starting the arbitrary waveform function.

2.4.3 getWaveformParameters

Summary
Queries the arbitrary waveform function parameters for a single controller channel.

Definition
```
LONG getWaveformParameters(LONG connectionID, LONG channel, PLONG enable, PLONG active, PLONG
dataLength, PLONG loopCyclesPerPoint, PLONG iterations, PLONG iterationCount, PLONG index)
```

Parameters

connectionID	Specifies the ID number of the connection.
channel	Specifies the controller channel number. Valid value range is 1 to 3.
enable	Pointer to a variable of type LONG which receives the enable parameter: 0 = disabled, 1 = infinite iterations, 2 = finite number of iterations.
active	Pointer to a variable of type LONG which receives the active parameter: 0 = not running, 1 = running.
dataLength	Pointer to a variable of type LONG which receives the length of the waveformData[] array that was written by the loadWaveform function.
loopCyclesPerPoint	Pointer to a variable of type LONG which receives the number 24µs control loop cycles per waveform point.
iterations	Pointer to a variable of type LONG which receives the number of iterations.
iterationCount	Pointer to a variable of type LONG which receives the iteration count. This value is decremented by the controller as iterations are completed.
index	Pointer to a variable of type LONG which receives the index value of the current arbitrary waveform function output point.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
When enable is a value of 1 or 2 the Arbitrary Waveform function output is summed with the Digital Position Command.

2.4.4 stopCh1Ch2Ch3Waveforms

Summary
Writes the command to stop the controller arbitrary waveform function for Ch1, Ch2, and Ch3.

Definition
```
LONG stopCh1Ch2Ch3Waveforms(LONG connectionID)
```

Parameters

connectionID	Specifies the ID number of the connection.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
This function is also used for single channel LC.400 controllers and two channel LC.402 controllers. This function sets Waveform Enable and Waveform Active to zero for Ch1, Ch2, and Ch3.

2.5 Spiral Scan

2.5.1 setSpiralParameters

Summary
Write the parameters for the controller internal spiral scan function.

Definition
```
LONG setSpiralParameters(LONG connectionID, LONG spiralAxis1, LONG spiralAxis2, FLOAT axis1Radius,
FLOAT axis2Radius, FLOAT lineSpacing, FLOAT revPerSec, FLOAT radiusStepVel, FLOAT dwellTime)
```

Parameters

connectionID	Specifies the ID number of the connection.
spiralAxis1	Specifies the controller channel number for Axis 1. Valid value range is 1 to 3.
spiralAxis2	Specifies the controller channel number for Axis 2. Valid value range is 1 to 3.
axis1Radius	Specifies the Axis 1 Radius in distance units.
axis2Radius	Specifies the Axis 2 Radius in distance units.
lineSpacing	Specifies the spacing between lines in distance units.
revPerSec	Specifies the revolutions per second (sine wave frequency).
radiusStepVel	Specifies the velocity of the step between the radius and the center position at the beginning or end of a scan frame depending on the scan mode specified when starting the scan.
dwellTime	Specifies the dwell time in seconds after the radius step completes.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
Increasing or decreasing mode (start at center or start at maximum radius) is chosen when calling the startSpiralScan function.

2.5.2 getSpiralParameters

Summary
Queries the parameters for the controller internal spiral scan function.

Definition
```
LONG getSpiralParameters(LONG connectionID, PLONG spiralAxis1, PLONG spiralAxis2, PFLOAT axis1Radius,
PFLOAT axis2Radius, PFLOAT lineSpacing, PFLOAT revPerSec, PFLOAT radiusStepVel, PFLOAT dwellTime)
```

Parameters

connectionID	Specifies the ID number of the connection.
spiralAxis1	Pointer to a variable of type LONG which receives the Spiral Axis 1 value.
spiralAxis2	Pointer to a variable of type LONG which receives the Spiral Axis 2 value.
axis1Radius	Pointer to a variable of type FLOAT which receives the Axis 1 Radius value.
axis2Radius	Pointer to a variable of type FLOAT which receives the Axis 2 Radius value.
lineSpacing	Pointer to a variable of type FLOAT which receives the Line Spacing value.
revPerSec	Pointer to a variable of type FLOAT which receives the Revolutions Per Second value.
radiusStepVel	Pointer to a variable of type FLOAT which receives the Radius Step Velocity value.

dwellTime	Pointer to a variable of type FLOAT which receives the Dwell Time value.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
See additional parameter details in the setSpiralParameters function.

2.5.3 setSpiralFrameParameters

Summary
Write the frame parameters for the controller internal spiral scan function.

Definition
```
LONG setSpiralFrameParameters(LONG connectionID, LONG frameAxis, FLOAT frameStepSize, LONG numFrames)
```

Parameters

connectionID	Specifies the ID number of the connection.
frameAxis	Specifies the controller channel number to step for each frame. Valid value range is 1 to 3.
frameStepSize	Specifies the distance to step the frame axis for each frame in distance units.
numFrames	Specifies the number of frames. The minimum value is 1.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
For two channel controllers frameAxis should be a value of 3, and frameStepSize should be a value of zero.

2.5.4 getSpiralFrameParameters

Summary
Queries the frame parameters for the controller internal spiral scan function.

Definition
```
LONG getSpiralFrameParameters(LONG connectionID, PLONG frameAxis, PFLOAT frameStepSize, PLONG
numFrames)
```

Parameters

connectionID	Specifies the ID number of the connection.
frameAxis	Pointer to a variable of type LONG which receives the Frame Axis value.
frameStepSize	Pointer to a variable of type FLOAT which receives the Frame Step Size value.
numFrames	Pointer to a variable of type FLOAT which receives the Number of Frames value.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
See additional parameter details in the setSpiralFrameParameters function.

2.5.5 startSpiralScan

Summary
Writes the command to start the controller internal spiral scan function.

Definition
```
LONG startSpiralScan(LONG connectionID, LONG startValue)
```

Parameters

connectionID	Specifies the ID number of the connection.
ch1Start	Specifies the spiral scan mode: 1 = decreasing amplitude (start at maximum radius), 2 = increasing amplitude (start at center).

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
Mode 2 "increasing amplitude" is recommended for most applications. The scan is centered at the Digital Position Command values for the corresponding axes.

2.5.6 getSpiralScanStatus

Summary
Queries status of the spiral scan to determine if the spiral scan is complete.

Definition
```
LONG getSpiralScanStatus(LONG connectionID, PLONG status)
```

Parameters

connectionID	Specifies the ID number of the connection.
status	Pointer to a variable of type LONG which receives the spiral scan status value. Status value of 1 = scan in progress, 0 = scan complete or stopped.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

2.5.7 stopSpiralScan

Summary
Writes the command to stop the controller internal spiral scan function.

Definition
```
LONG stopSpiralScan(LONG connectionID)
```

Parameters

connectionID	Specifies the ID number of the connection.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
When the spiral scan is stopped with this function, the spiral position command that is summed with the Digital Position Command is immediately removed and the stage steps back to the center.

2.6 Recording to Internal Buffers

2.6.1 setRecPointer

Summary
Sets the internal controller value to record in one of the eight recording buffers.

Definition
```
LONG setRecPointer(LONG connectionID, LONG recBufferIndex, LONG channel, LONG recNode)
```

Parameters

connectionID	Specifies the ID number of the connection.
recBufferIndex	Specifies the index of the buffer to record to. Valid value range is 0 to 7.
channel	Specifies the channel for the value to be recorded. Valid channel numbers are 1 to 3.
recNode	Enumeration value for the following: 1 = Sensor Monitor (read back in distance units by the getScaledRecBuffer function) 2 = Position Command (read back in distance units by the getScaledRecBuffer function) 3 = Control Loop Error (read back in distance units by the getScaledRecBuffer function) 4 = BNC Analog Input (read back in Volts by the getScaledRecBuffer function)

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
To record nothing in a buffer, both the channel and recNode parameters should be set to a value of 0.

2.6.2 startRec

Summary
Writes the command to start the controller internal recording function. The controller can record up to eight values simultaneously every 24 µs. The sample rate can be slowed to any multiple of 24 µs to record for longer periods of time.

Definition
```
LONG startRec(LONG connectionID, LONG numSamples, LONG loopCyclesPerSample)
```

Parameters

connectionID	Specifies the ID number of the connection.
numSamples	Specifies the number of samples to record. Maximum value is 83,333 to not overwrite the next buffer.
loopCyclesPerSample	Specifies the number of 24 µs control loop samples per sample.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
setRecPointer should be called prior to startRec to set what values are recorded in the eight buffers.

2.6.3 getRecStatus

Summary
Queries status of the controller internal recording function to determine if the recording is complete.

Definition
```
LONG getRecStatus(LONG connectionID, PLONG status, PLONG index)
```

Parameters

connectionID	Specifies the ID number of the connection.
status	Pointer to a variable of type LONG which receives the recording status value. Status value of 1 = recording in progress, 0 = recording complete.
index	Pointer to a variable of type LONG which receives the recording index value. The index indicates the progress of the recording. After the recording is completed the DSP recording function automatically sets the index to zero.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

2.6.4 getScaledRecBuffer

Summary
Queries the data stored in one of eight internal controller recording buffers.

Definition
```
LONG getScaledRecBuffer(LONG connectionID, LONG recBufferIndex, PFLOAT retBuffer[], LONG retBufferSize)
```

Parameters

connectionID	Specifies the ID number of the connection.
recBufferIndex	Specifies the index of the buffer to query. Valid value range is 0 to 7.
retBuffer[]	Pointer to an array of type FLOAT that receives the scaled internal recording buffer data.
retBufferSize	Specifies the size of the return buffer. In most cases the maximum size is 83,333.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
The data is scaled to distance units or Volts depending on the recNode specified in the setRecPointer function.

2.7 Servo

2.7.1 setServoState

Summary
Write the servo state (open loop or closed loop) for a single controller channel.

Definition
```
LONG setServoState(LONG connectionID, LONG channel, LONG servoState)
```

Parameters

connectionID	Specifies the ID number of the connection.
channel	Specifies the controller channel number. Valid value range is 1 to 3.
servoState	Specifies the servo state: 0 = Servo Off (open loop), 1 = Servo On (closed loop).

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
Caution should be used when setting the system to closed loop if the control loop tuning is in an unknown state. If the control loop settings don't match the mechanical resonances, the control loop can become unstable and cause the mechanical system to oscillate with a large amplitude.

2.7.2 getServoState

Summary
Queries the servo state (open loop or closed loop) for a single controller channel.

Definition
```
LONG getServoState(LONG connectionID, LONG channel, PLONG servoState)
```

Parameters

connectionID	Specifies the ID number of the connection.
channel	Specifies the controller channel number. Valid value range is 1 to 3.
range	Pointer to a variable of type LONG which receives the servo state: 0 = Servo Off (open loop), 1 = Servo On (closed loop).

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

2.8 Saving Configuration or Arbitrary Waveform

2.8.1 saveConfigToController

Summary
Write the command to save the current configuration to controller flash memory.

Definition
```
LONG saveConfigToController(LONG connectionID)
```

Parameters

connectionID	Specifies the ID number of the connection.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
After saving the configuration to controller flash memory, the controller will subsequently start with this configuration. Arbitrary waveform data is not saved with this command. For most applications the controller should start with the arbitrary waveform function disabled, so it is not important to save the arbitrary waveform data. If saving arbitrary waveform data is required, see the saveWaveformToController function.

2.8.2 getSaveConfigStatus

Summary
Queries the status of the saveConfigToController function.

Definition
```
LONG getSaveConfigStatus(LONG connectionID, LONG status)
```

Parameters

connectionID	Specifies the ID number of the connection.
status	Pointer to a variable of type LONG which receives the status: 0 = Save Complete, 1 = Save In Progress.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
The saveConfigToController function normally completes in several seconds depending on how many channels are present in the controller and how many digital filters are applied.

2.8.3 saveWaveformToController

Summary
Write the command to save the waveform data for a single channel to flash memory.

Definition
```
LONG saveWaveformToController(LONG connectionID, LONG channel)
```

Parameters

connectionID	Specifies the ID number of the connection.
channel	Specifies the controller channel number. Valid value range is 1 to 3.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
Wait for the current channel to complete the waveform data save prior to saving the next channel. The save complete status can be checked with the getSaveWaveformStatus function.

2.8.4 getSaveWaveformStatus

Summary
Queries the status of the controller saveWaveformToController function.

Definition
```
LONG getSaveWaveformStatus(LONG connectionID, LONG status)
```

Parameters

connectionID	Specifies the ID number of the connection.
status	Pointer to a variable of type LONG which receives the status: 0 = Save Complete, 1 = Save In Progress.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
The saveWaveformToController function can take on the order of 10 seconds to complete depending on how many channels are present in the controller and how many digital filters are applied.

2.9 Direct Memory Read/Write

2.9.1 setInteger

Summary
Write a 32 bit signed integer value directly to a controller memory address.

Definition
```
LONG setInteger(LONG connectionID, LONG address, LONG value)
```

Parameters

connectionID	Specifies the ID number of the connection.
address	Specifies the controller memory address.
value	Specifies the value to be written.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

2.9.2 getInteger

Summary
Queries a 32 bit signed integer value directly from a controller memory address.

Definition
```
LONG getInteger(LONG connectionID, LONG address, PLONG retValue)
```

Parameters

connectionID	Specifies the ID number of the connection.
address	Specifies the controller memory address.
retValue	Pointer to a variable of type LONG which receives the value.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

2.9.3 setDiscreteIntArray

Summary
Write multiple non-contiguous 32 bit signed integer values directly to controller memory addresses in a single communication packet so that the values are changed in controller memory as close to the same time as possible rather than having communication latency between individual setInteger function calls.

Definition
```
LONG setDiscreteIntArray(LONG connectionID, PLONG addresses[], LONG addressesLength, PLONG values[])
```

Parameters

connectionID	Specifies the ID number of the connection.
addresses[]	Pointer to an array of type LONG that contains a list of the controller memory addresses.
addressesLength	Specifies the length of the addresses array.
Values[]	Pointer to an array of type LONG that contains a list of the values. The length of the array must be the same as the addresses[] array.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
For contiguous memory addresses such as arbitrary waveform data, the setByteArray function has less communication overhead.

2.9.4 getDiscreteIntArray

Summary
Query multiple non-contiguous 32 bit signed integer values directly from controller memory addresses in a single communication packet so that the values are read from controller memory as close to the same time as possible rather than having communication latency between individual getInteger function calls.

Definition
```
LONG setDiscreteIntArray(LONG connectionID, PLONG addresses[], LONG addressesLength, PLONG retVals[])
```

Parameters

connectionID	Specifies the ID number of the connection.
addresses[]	Pointer to an array of type LONG that contains a list of the controller memory addresses.
addressLength	Specifies the length of the addresses array.
retVals[]	Pointer to an array of type LONG that receives the values. The length of the array must be the same as the addresses[] array.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
For contiguous memory addresses such as internal recording buffer data, the getByteArray function has less communication overhead.

2.9.5 setSingle

Summary
Write a 32 bit floating point value directly to a controller memory address.

Definition
```
LONG setSingle(LONG connectionID, LONG address, FLOAT value)
```

Parameters

connectionID	Specifies the ID number of the connection.
address	Specifies the controller memory address.
value	Specifies the value to be written.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

2.9.6 getSingle

Summary
Queries a 32 bit floating point value directly from a controller memory address.

Definition
```
LONG getSingle(LONG connectionID, LONG address, PFLOAT retValue)
```

Parameters

connectionID	Specifies the ID number of the connection.
address	Specifies the controller memory address.
retValue	Pointer to a variable of type FLOAT which receives the value.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

2.9.7 setDouble

Summary
Write a 64 bit floating point value directly to a controller memory address.

Definition
```
LONG setDouble(LONG connectionID, LONG address, DOUBLE value)
```

Parameters

connectionID	Specifies the ID number of the connection.
address	Specifies the controller memory address.
value	Specifies the value to be written.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

2.9.8 getDouble

Summary
Queries a 64 bit floating point value directly from a controller memory address.

Definition
```
LONG getDouble(LONG connectionID, LONG address, PDOUBLE retValue)
```

Parameters

connectionID	Specifies the ID number of the connection.
address	Specifies the controller memory address.
retValue	Pointer to a variable of type DOUBLE which receives the value.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

2.9.9 setByteArray

Summary
Write an array of bytes directly to contiguous controller memory beginning at a specified address.

Definition
```
LONG setByteArray(LONG connectionID, LONG startAddress, PUCHAR byteArray[], LONG byteArrayLength)
```

Parameters

connectionID	Specifies the ID number of the connection.
startAddress	Specifies the controller memory address to begin writing to.
byteArray[]	Pointer to an array of type UNSIGNED CHAR that contains the values.
byteArrayLength	Specifies the length of the byte array.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

2.9.10 getByteArray

Summary
Queries an array of bytes directly from contiguous controller memory beginning at a specified address.

Definition
```
LONG getByteArray(LONG connectionID, LONG startAddress, PUCHAR retArray[], LONG retArrayLength,
LONG maxBytesPerRead)
```

Parameters

connectionID	Specifies the ID number of the connection.
startAddress	Specifies the controller memory address to begin reading from.
retArray[]	Pointer to an array of type UNSIGNED CHAR that receives the values.
retArrayLength	Specifies the length of the byte array.
maxBytesPerRead	Specifies the maximum number of bytes to read per communication packet.

Return Value
Returns a value of zero if successful, otherwise an error code defined in Chapter 1.

Remarks
For USB connections the maxBytesPerRead value should be 8000.

3 Programming Examples

3.1 Visual Basic .NET 2010

3.1.1 LC400_Example_API

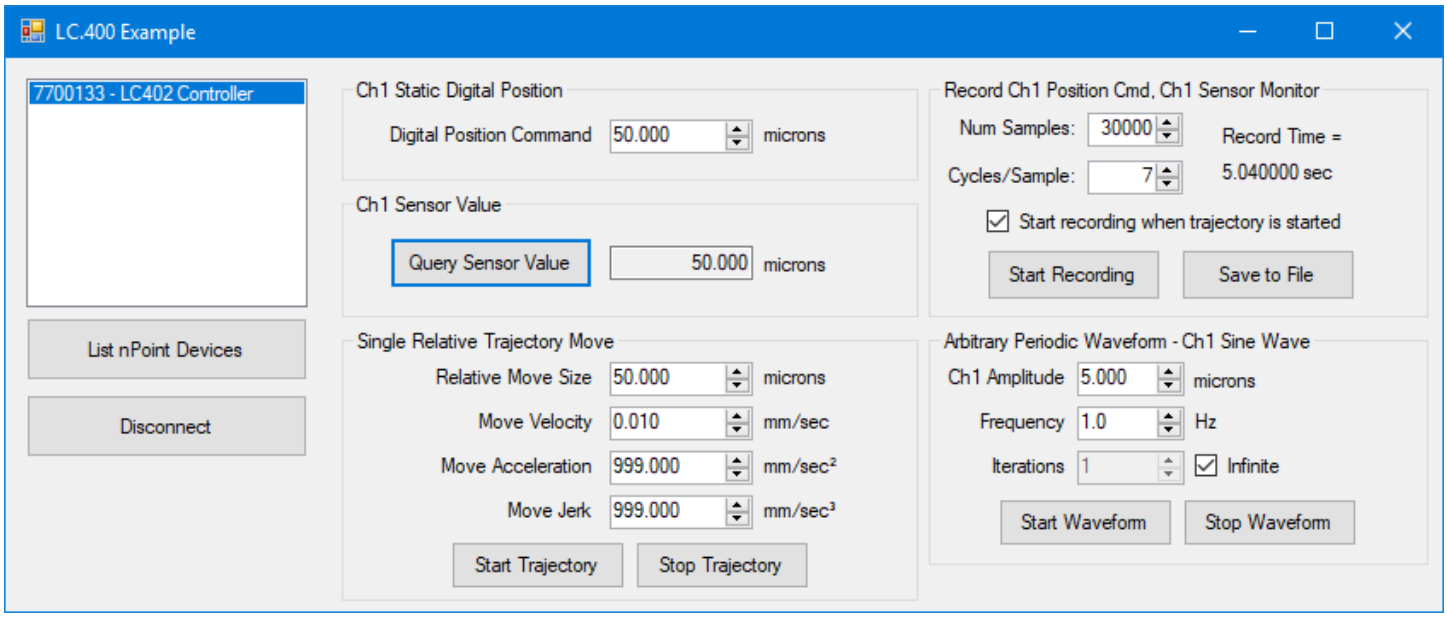

Figure 1 - LC400_Example_API user interface

The LC400_Example_API Visual Basic .NET 2010 example project demonstrates how to use the LC.400 API to perform the following tasks:

- List available controllers connected via USB displaying the serial number and description.
- Connect to a single controller selected from the list.
- Set the Ch1 Static Digital Position command.
- Query the Ch1 Sensor Monitor value.
- Perform a Ch1 single trajectory move relative to the current Static Digital Position Command.
- Simultaneously record the Ch1 Position Command and Ch1 Sensor Monitor values, automatically start the recording immediately prior to starting the Trajectory move, and save the recording buffer to a .CSV file on the PC.
- Load and start a periodic sine wave function with the Ch1 Arbitrary Waveform functionality that continues indefinitely or automatically stops after a specified number of iterations.

3.1.2 LC400_Example_SpiralScanAPI

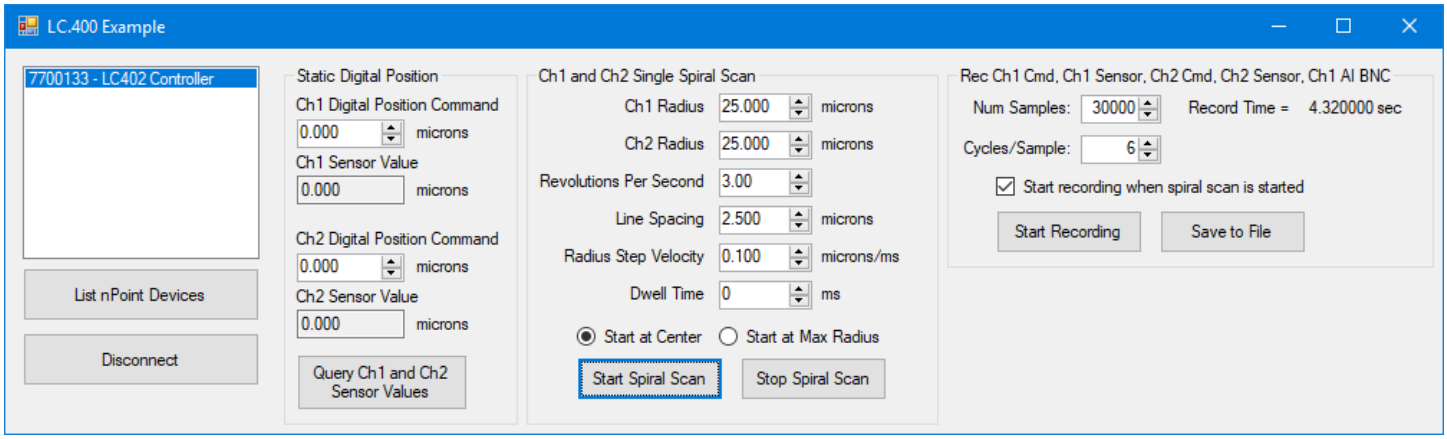

Figure 2 - LC400_Example_SpiralScanAPI user interface

The LC400_Example_SpiralScanAPI Visual Basic .NET 2010 example project demonstrates how to use the LC.400 API to perform the following tasks:

- List available controllers connected via USB displaying the serial number and description.
- Connect to a single controller selected from the list.
- Set the Ch1 and Ch2 Static Digital Position command values.
- Query the Ch1 and Ch2 Sensor Monitor values.
- Perform a 2 axis spiral scan with Ch1 and Ch2.
- Simultaneously record the Ch1 Position Command, Ch1 Sensor Monitor, Ch2 Position Command, Ch2 Sensor Monitor, and Ch1 BNC Analog Input voltage. The Ch1 BNC Analog Input voltage is recorded for alignment applications that connect an optical intensity signal to image intensity vs. stage position.
- Automatically start the recording immediately prior to starting the spiral scan, and save the recording buffer to a .CSV file on the PC.