CPL590 API Document

Lion Precision/MTA
St. Paul, Minnesota, USA
Telephone: 651-484-6544
www.lionprecision.com

Contents

1.0 Lion.Device.CPL590 ... 4
1.1 Constructor ... 4
1.2 Public Attributes ... 4
1.2.1 TypeDevice ... 4
1.2.2 TypeApi ... 5
1.3 Properties ... 5
1.3.1 StateApi ... 5
1.3.2 Running ... 5
1.3.3 Frequency ... 6
1.3.4 MaximumRpm ... 6
1.3.5 MaxSamplingRate ... 6
1.3.6 NoSpinSamplingRate ... 7
1.4 Methods ... 8
1.4.1 GetSystem ... 9
1.4.2 GetChassis ... 9
1.4.3 GetAvailableChannels ... 10
1.4.4 GetActiveChannels ... 12
1.4.5 GetOption ... 13
1.4.6 SetOption ... 14
1.4.7 Connect Channel ... 16
1.4.8 Connect List ... 17
1.4.9 ConnectAsync Channel ... 18
1.4.10 ConnectAsync List ... 19
1.4.11 Disconnect Channel ... 20
1.4.12 Disconnect List ... 21
1.4.13 Shutdown ... 22
1.4.14 Read ... 22
1.4.15 GetStream ... 23
1.4.16 GetDeviceInfo ... 24
1.4.17 GetCalibrations ... 25
1.4.18 GetCalibrationsAsync ... 25
1.4.19 GetDataStatistics ... 26
1.4.20 GetDataStatisticsAsync ... 27
1.4.21 GetDataStatisticsByChannel ... 28
1.4.22 GetDataStatisticsByChannelsAsync ... 28
1.4.23 SetSensitivityLevel ... 29
1.4.24 Dispose ... 30
2.0 Lion.Base ... 32
2.1 Properties ... 32
2.1.1 SystemConfiguration ... 32

2.1.2 ChassisConfiguration 36
2.1.3 ChannelConfiguration 39
2.1.4 DeviceInfo 47
2.1.5 AsyncState 47
2.1.6 AnalysisOptions 47
2.1.7 TransferOptions 48
2.1.8 Calibration 48
2.1.9 Measurement 49
2.2 Enumerations 50
2.2.1 ApiState 50
2.2.2 ApiType 50
2.2.3 DeviceMode 50
2.2.4 ChannelType 50
2.2.5 ChassisType 50
2.2.6 DataType 51
2.2.7 2.2.7........ SensitivityLevel 51
2.2.8 SamplingMode 51
2.2.9 TransferOptionName 52
2.2.10 AnalysisOptionName 52
3.0 Lion.Stream 53
3.1 Public Member Functions 53
3.1.1 BeginRead 54
3.1.2 CopyTo 55
3.1.3 CopyToAsync 56
3.1.4 EndRead 58
3.1.5 Flush 58
3.1.6 Read 59
3.1.7 ReadAsync 59
3.1.8 Dispose 60
4.0 CPL590 Data Receiving Examples 61
4.1 Manual Mode 61
4.2 Auto Mode 62

1.0 Lion.Device.CPL590

1.1 Constructor

Cpl590Api

This namespace provides access to the CPL59X Board for the 2U CPL590 System

Syntax (C#):

Cpl590Api ()

Assemblies:
Lion.Device.CPL590.dll

Example:

```
public static void Main()
{
     // Get Instance of API
     var apiHandle = Cpl590Api();

     // Get the System Configuration structure
     var system = apiHandle.GetSystem();
}
```

1.2 Public Attributes

TypeDevice	The type of device. Uses the DeviceMode structure. Default is DeviceMode.Cpl590.
TypeApi	The type of API. Uses the ApiType structure. Default for this API is ApiType.Device.

1.2.1 TypeDevice

The type of device that is accessed with this API (Elite, CPL590, SpindleCheck, etc).

Syntax (C#)

DeviceMode TypeDevice

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

1.2.2 TypeApi

The type of API.

Syntax (C#)

ApiType TypeApi

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

Remarks:

For this CPL590 API, this value is preset to ApiType.Device.

1.3 Properties

StateApi	The current state of the API
Running	The flag indicating whether the Data Receiving is Active.
Frequency	The Frequency (RPS) of the data buffer
MaximumRpm	The maximum RPM. Default is 15000_{10}.
MaxSamplingRate	The maximum sampling rate of the Device. Default is 67000_{10}.
NoSpinSamplingRate	The Sampling Rate to be set when the Spindle is NOT Rotating. Default is 67000_{10}.

1.3.1 StateApi

The current state of the API.

Syntax (C#):

ApiState StateApi

1.3.2 Running

Gets a value indicating the state of data transfer with the CPL591/2.

Syntax (C#)

bool Running

Assemblies: Lion.Device.CPL590.dll

Remarks:

After a successful data transfer is initiated, this flag will be set to true.

Example:

```
public static void DataHandler(CPL590Api apiHandle, int dataSize)
{
    // Get the System Configuration structure
    var system = apihandle.GetSystem();

    // Set the data size
    apihandle.SetOption(AnalysisOpttionName.ChannelBufferSize, dataSize);

    // Build test data buffer
    var testData = new double[channelCount, dataSize];

     //  Use the GetStream public method to return the LionStream.
     var lionStream = apiHandle.GetStream();

     // Perform a BeginRead function to start receiving data
     lionStream.BeginRead(testData, 0, dataSize, DataCallbackS, null);


     //  If Begin did not happen, error
     if (!apiHandle.Running)
    {
         Console.Write("Data Did NOT Begin!")
    }
}
```

1.3.3 Frequency

Gets the frequency (Hz) of the Input Data signal.

Syntax (C#):

double Frequency

Assemblies: Lion.Device.CPL590.dll

Remarks:

This value is actually the Revolutions Per Second detected on the Data Buffer with the Spindle Rotating. If the Spindle is not rating, this value is set to zero.

1.3.4 MaximumRpm

The Maximum RPM that this Device can send to the Spindle.

Syntax (C#):

double MaximumRpm

Assemblies: Lion.Device.CPL590.dll

Remarks:

This value is to be used the main application software, which will control the Spindle RPM for testing.

1.3.5 MaxSamplingRate

The maximum Sampling Rate (Samples per second) that this Device can handle.

Syntax (C#):

double MaxSamplingRate

Assemblies: Lion.Device.CPL590.dll

Remarks:

This value is to be used by the Lion.Core data processing routines when calculating the new Sampling rate for Auto Sampling mode operations.

1.3.6 NoSpinSamplingRate

The Sampling Rate (Samples per second) generated by this Device when it detects that the Spindle is not rotating.

Syntax (C#):

```
double NoSpinSamplingRate
```

Assemblies: Lion.Device.CPL590.dll

Remarks:

This value is to be used by the Lion.Core data processing routines when calculating the new Sampling rate for Auto Sampling mode operations.

1.4 Methods

GetSystem ()	Get the structure of the System
GetChassis ()	Get the array of available chassis.
GetAvailableChannels ()	Get the array of available channels (devices).
GetActiveChannels ()	Get the array of active channels (devices).
GetOption (*OptionsName*)	Get the options used for running Data transfer on all devices.
SetOption (*AnalysisOpttionName , object*)	Set the options used for running Data transfer on all devices.
Connect (*ChannelConfiguration*)	Opens a connection to a CPL59X Driver.
Connect (List< *ChannelConfiguration* >)	Opens a connection to a strong typed List of CPL59X Channels(Devices)
ConnectAsync (*ChannelConfiguration, CancellationToken*)	Opens a connection to a CPL59X as an asynchronous operation. Can be cancelled using the CancellationToken
ConnectAsync (List < *ChannelConfiguration* >, *CancellationToken*)	Opens a connection to a strong typed List of CPL59X Channels (Devices) as an asynchronous operation. Can be cancelled using the CancellationToken
Disconnect (*ChannelConfiguration*)	Closes a connection to a CPL59X Driver.
Disconnect (*List*< *ChannelConfiguration* >)	Closes a connection to an array of CPL59X Drivers.
Shutdown ()	Shut down (closes) all Active CPL59X Drivers
Read (*int*)	Reads Data from all Active CPL59X Drivers.
GetStream ()	Returns the Stream class used to receive data.
GetDeviceInfo ()	Get the CPL590 FTDI Chip device information.
GetCalibrations ()	Get the Calibration (TEDs) information from the CPL59X Driver.
GetCalibrationsAsync (*CancellationToken*)	Get the Calibration (TEDs) information from the CPL59X Driver as an asynchronous operation.
GetDataStatistics ()	Get the Statistics of the incoming Data (Min, Max, Average, etc.)
GetDataStatisticsAsync (*CancellationToken*)	Get the Statistics of the incoming Data (Min, Max, Average, etc.) as an asynchronous operation.

GetDataStatisticsByChannel (*int*)	Get the Statistics of the incoming Data (Min, Max, Average, etc.) by channel.
GetDataStatisticsByChannelsAsync(*int*, *CancellationToken*)	Get the Statistics of the incoming Data (Min, Max, Average, etc.) by channel as an asynchronous operation.
SetSensitivityLevel (*int*, SensitivityLevel)	Sets the Sensitivity level (probe range) of the CPL59X Driver.
Dispose ()	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

1.4.1 GetSystem

Get the structure of the CPL59X System.

Syntax (C#):

SystemConfiguration GetSystem ()

Assemblies:
Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:
None

Returns:

SystemConfiguration	The SystemConfiguration property.

Exceptions:

None

Remarks:
Call this method to retrieve the contents of the System structure. This structure contains the List of Chassis's in the System as well as the Data Transfer and Calculation settings. The System structure is built when the GetAvailableChannels method is run on the CPL590 API.

Example:

```
    public static void Main()
    {
            // Get Instance of API
            var apiHandle = Cpl590Api();

            // Get the System Configuration structure
            var system = apiHandle.GetSystem();

            // Display number of Chassis found in the System
            Console.Write(system.ChassisCount);
    }
```

1.4.2 GetChassis

Get the strong typed List of available chassis installed in the System.

Syntax (C#):

List<ChassisConfiguration> GetChassis ()

Assemblies:
Lion.Device.CPL590.dll, Lion.Base.dll, System.Collections.Generic.dll

Parameters:
None

Returns:

List< ChassisConfiguration>	The strong typed List of the ChassisConfiguration property.

Exceptions:
None

Remarks:
Call this method to retrieve the List array of the Chassis structure. For CPL590 One-Channel System, the count of the List matches the number of Channels (Devices) that are to be connected and run data transfer. This List is built when the GetAvailableChannels method is run on the CPL590 API.

Example:

```
public static void Main()
{
     // Get Instance of API
     var apiHandle = Cpl590Api();

     // Get the Chassis Configuration structure
     var chassis = apiHandle.GetChassis();

     // Display number of Chassis found in the System
     Console.Write(chassis.Count);
}
```

1.4.3 GetAvailableChannels

Get the strong typed List of available devices.

Syntax (C#):

List< ChannelConfiguration > GetAvailableChannels ()

Assemblies:
Lion.Device.CPL590.dll, Lion.Base.dll, System.Collections.Generic.dll

Parameters:
None

Returns:

List< ChannelConfiguration >	The strong typed List of the ChannelConfiguration property.

Exceptions:
None

Remarks:
Call this method to get a strong typed List of all CPL59X Channels (Devices) connected to the USB Bus of the PC. The size of the List matches the number of Devices that are to be connected and run data transfer. For CPL590 One Channel System,

the count of the List matches the number of Channels (Devices) that are to be connected and run data transfer. The '*IsEnable*' item of the property will be set to '*false*' for each Device.
Calling this method will build the System and Chassis structures based on the devices found.

Example:

```
    public static void Main()
    {
            // Get Instance of API
            var apiHandle = Cpl590Api();

            // Get array of Devices found on USB Bus
            List<ChannelConfiguration> deviceList = apiHandle.GetAvailableChannels();

            // Cycle through the array and display the Serial Number on the Console
            foreach (ChannelConfiguration driver in deviceList)
            {
                    // Get Serial Numbers from array
                    Console.Write(driver.SerialNumber);
            }
    }
```

1.4.4 GetActiveChannels

Get the strong typed List of active Channels (devices). Active devices are devices that have been connected (opened) and ready to run data transfer.

Syntax (C#):

List< ChannelConfiguration > GetActiveChannels()

Assemblies:

Lion.Device.CPL590.dll, Lion.Base.dll

Parameters: None

Returns:

List< ChannelConfiguration >	The strong typed List of the ChannelConfiguration property.

Exceptions: None

Remarks:
Call this method to get an array of all CPL59X Channels (Devices) that had the 'Connect' function successfully run on the device. The '*IsEnable*' and the '*IsReady*' items of the *ChannelConfiguration* property should be set to '*true*' for each Device. If the operator has not run the 'Connect' function, the array will be empty.

Example:
```
public static void Main()
{
    // Get API
    var apiHandle = CPL590Api();

    // Get List of Devices on USB Bus
    List<ChannelConfiguration> deviceList = apiHandle.GetAvailableChannels();

    // Cycle through the list and enable all channels
    foreach (var driver in deviceList)
    {
        driver.IsEnabled = true;
    }

    // Cycle through the list and open all channels
    foreach (var driver in deviceList)
    {
       // Connect to Device
       apiHandle.Connect(driver);

       // Check Connected flag
       if (driver.IsConnected == false)
       {
           Console.Write("Connect Failed.");
       }
    }

    // Get Actives List
    List<ChannelConfiguration> activeList = apiHandle.GetActiveChannels();

    // Cycle through the list and display the Serial Number on the Console
    foreach (var driver in activeList)
    {
          // Get Serial Numbers from List
          Console.Write(driver.SerialNumber);
    }
}
```

1.4.5 GetOption

Returns the value of a specified configuration option, for the System or Channel, represented as an object.

Syntax (C#):

object GetOption (TransferOptionName)	Returns the value of a specified TransferOptions, represented as an object.
object GetOption (AnalysisOptionName)	Returns the value of a specified AnalyzeOptions, represented as an object.
object GetOption (EncoderOptionName)	Returns the value of a specified EncoderOptions, represented as an object. ***This feature is not used in current CPL59X Systems.***
object GetOption (IndexOptionName)	Returns the value of a specified IndexOptions, represented as an object. ***This feature is not used in current CPL59X Systems.***
object GetOption (TemperatureOptionName)	Returns the value of a specified TemperatureOptions, represented as an object. ***This feature is not used in current CPL59X Systems.***

object GetOption (TransferOptionName optionName)

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:

optionName	One of the TransferOptionName values.

Returns:

object	An object that represents the value of the option.

Exceptions:

InvalidOperationException	Illegal Options Name.
Exception	

Remarks:

TransferOptions determine the behavior for receiving Data such as number of channels, size of data per channel, etc.

Example:

```
public void Execute(CPL590Api apiHandle, double[,] inputData)
{

     // Get the Number of Channels (Rows) from the Data buffer
     var channelCount = inputData.GetLength(0);

     // Get the configured channel count
     var numberChannels = (double)apiHandle.GetOption(TransferOptionName.DataChannelCount);

     // Display error if number of channels do not match
     if (numberChannels != channelCount)
         Console.Write($"Fail! Channel counts do not match!");
}
```

object GetOption (AnalysisOptionName optionName)

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:

optionName	One of the AnalyzeOptions values.

Returns:

object	An object that represents the value of the option.

Exceptions:

InvalidOperationException	Illegal Options Name.
Exception	

Remarks:

AnalyzeOptions determine the behavior for analyzing the received Data such as sampling rate, number of revolutions, etc.

Example:

```
public void Execute(CPL590Api apiHandle, double[,] inputData)
{
     // Get the configured channel count
     var setSamplingRate = (double)apiHandle.GetOption(TransferOptionName.DataChannelCount);

     // Display error if number of channels do not match
     if (numberChannels != channelCount)
         Console.Write($"Fail! Channel counts do not match!");
}
```

1.4.6 SetOption

Sets the specified configuration option, for the System or Channel, represented as an object.

Syntax (C#):

SetOption (TransferOptionName, object optionValue)	Sets the value of a specified TransferOptions, represented as an object.
SetOption (AnalysisOptionName, object optionValue)	Sets the value of a specified AnalyzeOptions, represented as an object.
SetOption (EncoderOptionName, object optionValue)	Sets the value of a specified EncoderOptions, represented as an object. ***This feature is not used in current CPL59X Systems.***
SetOption (IndexOptionName, object optionValue)	Sets the value of a specified IndexOptions, represented as an object. ***This feature is not used in current CPL59X Systems.***
SetOption (TemperatureOptionName, object optionValue)	Sets the value of a specified TemperatureOptions, represented as an object. ***This feature is not used in current CPL59X Systems.***

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:

optionName	One of the **AnalysisOptionName** values.
serialNumber	The serial number of the Channel's configuration. Use empty string if setting Data Transfer or Analysis options.
optionValue	The value of the option.

Returns:

Exceptions:

InvalidOperationException	Illegal Options Name.
Exception	

Remarks:

Example:

```
public static void Main()
{
    // Get API
    var apiHandle = CPL590Api();

    // Set the Sampling Rate to No Spin
    apihandle.SetOption(TransferOptionName.CalculatedSamplingRate,
           apihandle.NoSpinSamplingRate);

    // Set the Auto Sampling mode
    apihandle.SetOption(AnalysisOptionName.SamplingMode, SamplingMode.Auto);

}
```

1.4.7 Connect Channel

Opens a connection to a CPL59X Channel(Device)

Syntax (C#):

void Connect (ChannelConfiguration *device*)

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:

device	The CPL59X Channel (Device) to which you intend to connect. The structure of the device is ChannelConfiguration.

Returns:

Exceptions:

ArgumentNullException	deviceIdentifier
InvalidOperationException	No Active CPL59X Drivers found.

Remarks:

Call this method to establish a USB connection to the specified CPL59X Channel (Device) designated by the *device.* This *device* is derived from the ChannelConfiguration property and contains the ID's of both the A and B side of the FT4222 chip on the Driver board.

Example:

```
public static void Main()
{
    // Get API
    var apiHandle = CPL590Api();

    // Get List of Devices on USB Bus
    var deviceList = apiHandle.GetAvailableChannels();

    // Cycle through the list and enable all channels
    foreach (var driver in deviceList)
    {
        driver.IsEnabled = true;
    }

    // Cycle through the list and enable all channels
    foreach (var driver in deviceList)
    {
       // Connect to Device
       apiHandle.Connect(driver);

       // Check Connected flag
       if (driver.IsConnected == false)
       {
           Console.Write("Connect Failed.");
       }
    }
}
```

1.4.8 Connect List

Opens a connection to a strong typed List of CPL59X Channels(Devices)

Syntax (C#):

void Connect (List< ChannelConfiguration > deviceList)

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:

deviceList	The strong typed List of CPL59X Channels (Devices) to which you intend to connect. The structure of the device is ChannelConfiguration.

Returns:

Exceptions:

ArgumentNullException	deviceIdentifier
InvalidOperationException	No Active CPL59X Drivers found.

Remarks:

Call this method to establish a USB connection to the specified CPL59X Channels (Devices) designated by the *device*. This *device* is derived from the ChannelConfiguration property and contains the ID's of both the A and B side of the FT4222 chip on the Driver board.

Example:

```
public static void Main()
{
    // Get API
    var apiHandle = CPL590Api();

    // Get List of Devices on USB Bus
    var deviceList = apiHandle.GetAvailableChannels();

    // Cycle through the list and enable all channels
    foreach (var driver in deviceList)
    {
        driver.IsEnabled = true;
    }

    // Connect to Devices and enable them
    apiHandle.Connect(deviceList);

    // Cycle through the list and check channels
    foreach (var driver in deviceList)
    {
       // Check Connected flag
       if (driver.IsConnected == false)
       {
           Console.Write("Connect Failed.");
       }
    }
}
```

1.4.9 ConnectAsync Channel

Opens a connection to a CPL59X Channel(Device) as an asynchronous operation.

Syntax (C#):

void Connect (ChannelConfiguration device, *CancellationToken* cancellationToken)

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:

device	The CPL59X Channel (Device) to which you intend to connect. The structure of the device is *ChannelConfiguration*.
cancellationToken	The Cancellation Token.

Returns:

Exceptions:

ArgumentNullException	deviceIdentifier
InvalidOperationException	No Active CPL59X Drivers found.

Remarks:

Call this method to establish a USB connection to the specified CPL59X Channel (Device) designated by the *device*. This *device* is derived from the ChannelConfiguration property and contains the ID's of both the A and B side of the FT4222 chip on the Driver board.

Example:

```
public static void Main()
{
    // Get API
    var apiHandle = CPL590Api();

    // Get Async cancellation token
    var _taskCancel = new CancellationTokenSource();

    // Get List of Devices on USB Bus
    var deviceList = apiHandle.GetAvailableChannels();

    // Cycle through the list and enable all channels
    foreach (var driver in deviceList)
    {
        driver.IsEnabled = true;
    }

    // Cycle through the list and enable all channels
    foreach (var driver in deviceList)
    {
       // Connect to Device
       apiHandle.ConnectAsync(driver, _taskCancel.Token).Wait(_taskCancel.Token);

       // Check Connected flag
       if (driver.IsConnected == false)
       {
           Console.Write("Connect Failed.");
       }
    }
}
```

1.4.10 ConnectAsync List

Opens a connection to a strong typed List of CPL59X Channels(Devices) as an asynchronous operation.

Syntax (C#):

void ConnectAsync (List< ChannelConfiguration > deviceList)

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:

deviceList	The strong typed List of CPL59X Channels (Devices) to which you intend to connect. The structure of the device is ChannelConfiguration.

Returns:

Exceptions:

ArgumentNullException	deviceIdentifier
InvalidOperationException	No Active CPL59X Drivers found.

Remarks:

Call this method to establish a USB connection to the specified CPL59X Channels (Devices) designated by the *device*. This *device* is derived from the ChannelConfiguration property and contains the ID's of both the A and B side of the FT4222 chip on the Driver board.

Example:

```
public static void Main()
{
    // Get API
    var apiHandle = CPL590Api();

    // Get Async cancellation token
    var _taskCancel = new CancellationTokenSource();

    // Get List of Devices on USB Bus
    var deviceList = apiHandle.GetAvailableChannels();

    // Cycle through the list and enable all channels
    foreach (var driver in deviceList)
    {
        driver.IsEnabled = true;
    }

    // Connect to Devices
    apiHandle.ConnectAsync(deviceList, _taskCancel.Token).Wait(_taskCancel.Token);

    // Cycle through the list and check channels
    foreach (var driver in deviceList)
    {
       // Check Connected flag
       if (driver.IsConnected == false)
       {
           Console.Write("Connect Failed.");
       }
    }
}
```

1.4.11 Disconnect Channel

Closes the connection to a CPL59X Channel(Device)

Syntax (C#):

void Disconnect (ChannelConfiguration device)

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:

device	The CPL59X Channel (Device) to which you intend to disconnect. The structure of the device is ChannelConfiguration.

Returns:

Exceptions:

ArgumentNullException	deviceIdentifier
InvalidOperationException	No Active CPL59X Drivers found.

Remarks:

Call this method to establish a USB connection to the specified CPL59X Channel (Device) designated by the *device*. This *device* is derived from the ChannelConfiguration property and contains the ID's of both the A and B side of the FT4222 chip on the Driver board.

Example:

```
public static void Close(CPL590Api apiHandle)
{
    // Get List of active Devices
    var deviceList = apiHandle.GetActiveChannels();

    // Cycle through the list and close all channels
    foreach (var driver in deviceList)
    {
       // Disconnect from Device
       apiHandle.Disconnect(driver);

       // Check Connected flag
       if (driver.IsConnected)
       {
           Console.Write("Disconnect Failed.");
       }
    }
}
```

1.4.12 Disconnect List

Closes the connection to a strong typed List of CPL59X Channels(Devices)

Syntax (C#):

void Disconnect (ChannelConfiguration device)

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:

device	The CPL59X Channel (Device) to which you intend to disconnect. The structure of the device is ChannelConfiguration.

Returns:

Exceptions:

ArgumentNullException	deviceIdentifier
InvalidOperationException	No Active CPL59X Drivers found.

Remarks:

Call this method to establish a USB connection to the specified CPL59X Channel (Device) designated by the *device.* This *device* is derived from the ChannelConfiguration property and contains the ID’s of both the A and B side of the FT4222 chip on the Driver board.

Example:

```
public static void Close(CPL590Api apiHandle)
{
    // Get List of active Devices
    var deviceList = apiHandle.GetActiveChannels();

    // Disconnect from Devices
    apiHandle.Disconnect(deviceList);
}
```

1.4.13 Shutdown

Shut down all Active CPL59X Drivers

Syntax (C#):

void Shutdown()

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:

Returns:

Exceptions:

ArgumentNullException	deviceIdentifier
InvalidOperationException	No Active CPL59X Drivers found.

Remarks:

Call this method to immediately disconnect all active Channels (Devices).

Example:

```
public static void Close(CPL590Api apiHandle)
{
    // Disconnect from all active Devices
    apiHandle.Shutdown();
}
```

1.4.14 Read

Reads data from all Active CPL59X Drivers and stores it to a Two-Dimensional 16 bit word jagged array.

Syntax (C#):

void Read (int dataSize)

Assemblies:

Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:

dataSize	Size of the data.

Returns:

Two-Dimensional Buffer of Data words received.

Exceptions:

InvalidOperationException	**Device** must be Connected.
ArgumentNullException	Data size must be at least 2K.

Remarks:

This method reads data into buffer and returns the number of words successfully read. The Read operation reads as much data as is available, up to the number of words specified by the size parameter. The resultant two-dimensional jagged array is grouped by channel. The first pointer in the jagged array is the channel number (minus one). The second pointer is the data word for the channel.

Example:

```
public static void ReceiveData( CPL590Api apiHandle)
{
    // Get Actives List (Devices that are enabled and connected)
    List<ChannelConfiguration> activeList = apiHandle.GetActiveChannels();

    // Read 4096 words from each device in the list
    double[,] inputData = apiHandle.Read(4096);

    // Display the contents of the 2nd channel's data word at location 140
    Console.Write(inputData[1, 140]);

}
```

1.4.15 GetStream

Returns the Stream class used to receive data.

Syntax (C#):

LionStream GetStream()

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll, Lion.Core.dll

Parameters:

Returns:

LionStream	The underlying *LionStream*.

Exceptions:

InvalidOperationException	Device: *SerialNumber* NOT Connected.

Remarks:

The GetStream method returns a class named LionStream which you can use to receive data. The LionStream class inherits from the Stream class, which provides a rich collection of methods and properties used to facilitate network communications.

You must call the Connect method first, or the GetStream method will throw an InvalidOperationException. This method passes the location of the Read function, to the LionStream class, which will execute when data is received.

Example:

```
public int ProduceWorkItems(CPL590Api apiHandle, ref double[,] mainData)
{
       // Get the Cancel Task flag
       var taskCancel = new CancellationTokenSource();

       // Use the GetStream public method to return the LionStream.
       var lionStream = apiHandle.GetStream();

       // Read the current 2D Data Frame from the Device
       var wordsRead = lionStream.ReadAsync(mainData, 8096, taskCancel.Token).Result;

       // Return the number of words received from the Device
       return wordsRead;
}
```

1.4.16 GetDeviceInfo

Read the CPL590 FTDI Chip device information.

Syntax (C#):

DeviceInfo[,] GetDeviceInfo()

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:

Returns:

DeviceInfo	The Device Information defined in DeviceInfoclass

Exceptions:

InvalidOperationException	**CPL590** Driver must be setup first.

Remarks:

Call this method to read the internal information of the FTDI 4222 Chips on the CPL59X Channel (Driver). The resultant data is a Two-Dimensional jagged array containing the information of both A and B sides of the FTDI 4222 Chip. The first pointer in the jagged array is the channel number (minus one). The second pointer is the side (A) or (B) of the chip. The (A) side is selected using 0 and the (B) side is selected using 1.

Example:

```
public static void Main()
{
    // Get API
    var apiHandle = CPL590Api();

    // Get Information of the Device’s FTDI Chips
    DeviceInfo[,] devInfo = apiHandle.GetDeviceInfo();

    // Display the Serial Number of the 2nd channel’s ‘B’ side of the FTDI Chip
    Console.Write(devInfo [1, 1].SerialNumber);
}
```

1.4.17 GetCalibrations

Gets the calibration (TEDs) data from the Channel Driver.

Syntax (C#)

Calibration[] GetCalibrations()

Assemblies:

Lion.Device.CPL590.dll, Lion.base.dll

Returns:

An array of TEDs data structures, one for each channel in the System.

Exceptions:

InvalidOperationException	Device NOT Connected.
DeviceErrorException	

Remarks:

The TED's (Calibration) information is stored on CPL59X Channel (Driver) board. It is highly recommended to run this function before data transfer in order to have the proper values for calculating displacement from the raw counts sent from the hardware driver.

Example:

```
  public Calibration[] ReadTEDs(CPL590Api apiHandle)
  {
      // Get the list of Active CPL59X Drivers
      var deviceList = apiHandle.GetActiveChannels();
      if (deviceList == null)
             throw new InvalidOperationException($"No Active CPL59X Drivers found.");

      // Set the Driver Sensitivity Range
      apiHandle.SetSensitivityLevel(SensitivityLevel.R1);

      // Read Calibration (TEDs) Information
      var tedsData = apiHandle.GetCalibrations();

      return tedsData;
}
```

1.4.18 GetCalibrationsAsync

Gets the calibration (TEDs) data from the Channel Driver as an asynchronous operation.

Syntax (C#)

async Task<Calibration[]> GetCalibrationsAsync (CancellationToken cancellationToken)

Assemblies:

Lion.Device.CPL590.dll, Lion.base.dll

Parameters:

cancellationToken	The Cancellation Token.

Returns:

The TEDs data stored on the CPL59X Channel (Driver). An array of TEDs data structures, one for each sensitivity range for the driver.

Exceptions:

InvalidOperationException	Device NOT Connected.
DeviceErrorException	

Remarks:

The TED's (Calibration) information is stored on CPL59X Channel (Driver) board. It is highly recommended to run this function before data transfer in order to have the proper values for calculating displacement from the raw counts sent from the hardware driver.

Example:

```
  public Calibration[] ReadTEDs(CPL590Api apiHandle)
  {
      // Get the list of Active CPL59X Drivers
      var deviceList = apiHandle.GetActiveChannels();
      if (deviceList == null)
             throw new InvalidOperationException($"No Active CPL59X Drivers found.");

      // Set the Driver Sensitivity Range
      apiHandle.SetSensitivityLevel(SensitivityLevel.R1);

      // Get Async cancellation token
      var taskCancel = new CancellationTokenSource();

      // Get Configuration (TEDs) data from the Driver
      var tedsData = apiHandle.GetCalibrationsAsync(taskCancel.Token).Result;

      return tedsData;
}
```

1.4.19 GetDataStatistics

Gets the Measurement information (Statistics) of the Receiving Data Buffers from the CPL59X Channels (Devices).

Syntax (C#):

Measurement [] GetDataStatistics()

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:

Returns:

Measurement []	The array of Measurement information for the last data block received from each CPL59X Driver.

Exceptions:

InvalidOperationException	No Active CPL59X Drivers found. Or Device NOT Connected.

Remarks:

The Measurement information consists of the Maximum value, the Minimum value, the Average value, and the Peak-to-Peak value of the receiving data block.

Example:

```
public void DisplayStatistics(CPL590Api apiHandle)
{
    // Read the Measurement data for all Drivers
    Measurement[] measData = _apiHandle.GetDataStatistics();

    // Display the Average value of the data block for the 2nd channel
    Console.Write(measData[1].Average);

}
```

1.4.20 GetDataStatisticsAsync

Gets the Measurement information (Statistics) of the Receiving Data Buffers from the CPL59X Channels (Devices) as an asynchronous operation.

Syntax (C#):

async Task< Measurement []> GetDataStatisticsAsync (CancellationToken cancellationToken)

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:

cancellationToken	The Cancellation Token.

Returns:

Measurement []	The array of Measurement information for the last data block received from each CPL59X Driver.

Exceptions:

InvalidOperationException	No Active CPL59X Drivers found. Or Device NOT Connected.

Remarks:

The Measurement information consists of the Maximum value, the Minimum value, the Average value, and the Peak-to-Peak value of the receiving data block.

Example:

```
public void DisplayStatistics(CPL590Api apiHandle)
{
    // Get Async cancellation token
    var taskCancel = new CancellationTokenSource();

    // Read the Measurement data for all Drivers
    Measurement[] measData = apiHandle.GetDataStatisticsAsync(taskCancel.Token).Result;

    // Display the Average value of the data block for the 2nd channel
    Console.Write(measData[1].Average);

}
```

1.4.21 GetDataStatisticsByChannel

Gets the Measurement information (Statistics) of the Receiving Data Buffers from the CPL59X Channels (Devices) by Channel number

Syntax (C#):

Measurement GetDataStatisticsByChannel(channel)

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:

channel	*The channel number of the CPL59X Driver.*

Returns:

Measurement	The Measurement information for the last data block received from the CPL59X Driver selected by channel number.

Exceptions:

InvalidOperationException	No Active CPL59X Drivers found. Or Device NOT Connected.

Remarks:

The *Measurement* information consists of the Maximum value, the Minimum value, the Average value, and the Peak-to-Peak value of the receiving data block.

Example:

```
public void DisplayStatistics(CPL590Api apiHandle, int channelNumber)
{
    // Read the Measurement data for selected channel
    Measurement measData = _apiHandle.GetDataStatisticsByChannel(channelNumber);

    // Display the Peak-to-Peak value of the data block
    Console.Write(measData.P2P);

}
```

1.4.22 GetDataStatisticsByChannelsAsync

Gets the *Measurement* information (Statistics) of the Receiving Data Buffers from the CPL59X Channels (Devices) by Channel number

Syntax (C#):

Measurement GetDataStatisticsByChannelAsync(channel, CancellationToken cancellationToken)

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:

channel	The channel number of the CPL59X Driver.
cancellationToken	The Cancellation Token.

Returns:

Measurement	The Measurement information for the last data block received from the CPL59X Driver selected by channel number.

Exceptions:

InvalidOperationException	No Active CPL59X Drivers found. Or Device NOT Connected.

Remarks:

The *Measurement* information consists of the Maximum value, the Minimum value, the Average value, and the Peak-to-Peak value of the receiving data block.

Example:

```
public void DisplayStatistics(CPL590Api apiHandle, int channelNumber)
{
    // Get Async cancellation token
    var taskCancel = new CancellationTokenSource();

    // Read the Measurement data for all Drivers
    Measurement[] measData = apiHandle.GetDataStatisticsAsync(taskCancel.Token).Result;

    // Display the Peak-to-Peak value of the data block
    Console.Write(measData.P2P);

}
```

1.4.23 SetSensitivityLevel

Sets the calibration range sensitivity levels on all active CPL592's in the system.

Syntax (C#)

int SetSensitivityLevel(SensitivityLevel sensitivityRange);

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:

sensitivityRange	The Sensitivity level. See *SensitivityLevel* property for structure.

Remarks:

The CPL592 provides two sensitivity calibrations (Range 1 and Range 2) for one probe. Refer to the calibration sheets for specific calibration information. Select the desired sensitivity with this switch. Switching sensitivities will usually require repositioning the probe. The CPL591 only has one range and will ignore this function.

Exceptions:

InvalidOperationException	**Device** must be Connected.
ArgumentException	Only valid on Channel 0. *or* Sensitivity Range Level must be 0 or 1.
DeviceErrorException	The FT4222 Chipset driver returned an error.
Exception	

Example:

```
public static void Main()
{
    // Get API
    var apiHandle = CPL590Api();

    // Get List of Devices on USB Bus
    var deviceList = apiHandle.GetAvailableChannels();

    // Cycle through the list and enable all channels
    foreach (var driver in deviceList)
    {
       // Connect to Device
       apiHandle.Connect(driver);

       // Check Connected flag
       if (driver.IsConnected == false)
       {
           Console.Write("Connect Failed.");
       }
    }

    // Check for High Sensitivity range in parameters
    var sensitivityRange = SensitivityLevel.R1;

    // Set the Driver Sensitivity Range
    apiHandle.SetSensitivityLevel(SensitivityLevel.R1);

}
```

1.4.24 Dispose

Releases unmanaged and - optionally - managed resources.

Syntax (C#)

void Dispose();

Assemblies: Lion.Device.CPL590.dll, Lion.Base.dll

Parameters:

Remarks:

The *Dispose* method performs all object cleanup, so the garbage collector no longer needs to call the objects' *Object.Finalize* override. Therefore, the call to the *SuppressFinalize* method prevents the garbage collector from running the *finalizer*. If the type has no *finalizer*, the call to *GC.SuppressFinalize* has no effect. Note that the actual work of releasing unmanaged resources is performed by the second overload of the Dispose method.

Exceptions:

Example:

```
public static void Close(CPL590Api apiHandle)
{
    // Dispose and release the API
    apiHandle.Dispose();
}
```

2.0 Lion.Base

2.1 Properties

SystemConfiguration	The structure defining the characteristics of the System
ChassisConfiguration	The structure defining the characteristics of the Chassis in the System.
ChannelConfiguration	The structure defining the characteristics of the Channels (Drivers) in the Chassis.
DeviceInfo	The internal information of the FTDI 4222 Chips on the CPL59X Channel (Driver).
AnalyzeOptions	The options for Data Calculations.
TransferOptions	The options for Data Transfer.
AsyncState	A user-defined object that qualifies or contains information about an asynchronous operation.
Calibration	The Calibration (TEDs) data for the device Channel.
Measurement	The Measurement values (Statistics) of the incoming Data.

2.1.1 SystemConfiguration

SystemConfiguration	Initializes a new instance of the SystemConfiguration class.
Add	Adds the Chassis to System Configuration.
Get	Gets the specified Chassis Configuration
ChassisCount	The number of Chassis in this System
ChassisList	The list of Chassis in this System.
DataTransferOptions	The data transfer settings.
DataAnalyzeOptions	The data analyze settings.

A. SystemConfiguration

Initializes a new instance of the SystemConfiguration class.

Syntax (C#)

void SystemConfiguration ()

Assemblies:

Lion.Base.dll

Remarks:

This method is automatically called when the GetAvailableChannels method is run on the API and the API has found Devices on the USB Bus. It does not have to be called by the application.

B. Add

Adds the Chassis to System Configuration.

Syntax (C#)

void Add (ChassisConfiguration item)

Assemblies:

Lion.Base.dll

Remarks:

This method is automatically called when the GetAvailableChannels method is run on the API and the API has found Devices on the USB Bus. On the CPL59X One-Channel System, there will be a different Chassis configuration added for each Device found. It does not have to be called by the application.

C. Get

Get (string SerialNumber)	Gets the specified Chassis Configuration by the serial number of the corresponding Channel (Driver)
Get (int ChannelNumber)	Gets the specified Chassis Configuration by the channel number

Syntax (C#)

ChassisConfiguration Get (string SerialNumber)

ChassisConfiguration Get (int ChannelNumber)

Assemblies: Lion.Base.dll

Remarks:

This method is called when the application needs to retrieve the configuration for one of the Chassis's installed the System. The Chassis selected will be the one that has a Channel with the corresponding Serial Number or Channel Number. If the Serial Number string is empty, or the Channel Number is -1, then the first Chassis in the list will be retrieved.

Example:

```
public static int GetChassisNumber(CPL590Api apiHandle, string serialNumber)
{
    // Get the System Configuration structure
    var system = apihandle.GetSystem();

    // Get the desired Chassis Configuration structure
    var chassis = system?.Get(serialNumber);

     //  Return with the number of this Chassis.
     return chassis.ChassisNumber;
}
```

D. ChassisCount

The number of Chassis in this System.

Syntax (C#)

int ChassisCount

Assemblies:

Lion.Base.dll

Remarks:

This property will give the application the number of Chassis's found in the System when the GetAvailableChannels method is run on the API and the API has found Devices on the USB Bus. On the CPL59X One-Channel System, the will be a different Chassis configuration added for each Device found.

Example:

```
public static int GetChassisCount(CPL590Api apiHandle)
{
    // Get the System Configuration structure
    var system = apihandle.GetSystem();

     //  Return with the number of Chassis.
     return system.ChassisCount;
}
```

E. ChassisList

The list of Chassis in this System.

Syntax (C#)

List<*ChassisConfiguration*> ChassisList

Assemblies:

Lion.Base.dll

Remarks:

This property will give the application the strong typed list of Chassis's found in the System when the GetAvailableChannels method was run on the API and the API has found Devices on the USB Bus. On the CPL59X One-Channel System, the will be a different Chassis configuration added for each Device found.

Example:

```
public static int GetChassisCount(CPL590Api apiHandle)
{
    // Get the System Configuration structure
    var system = apihandle.GetSystem();

     //  Return with the number of Chassis.
     return system.ChassisCount;
}
```

F. DataTransferOptions

The options for Data Transfer.

Syntax (C#)

TransferOptions DataTransferOptions

Assemblies:

Lion.Base.dll

Remarks:

This property will give the application the ability to set/get the parameters (options) for running data transfer (read) from the Channels.

Example:

```
public static void SetOptions(CPL590Api apiHandle, int dataSize, double sampleRate)
{
    // Get the System Configuration structure
    var system = apihandle.GetSystem();

    // Set new options
    var system.DataTransferOptions = new TransferOptions ()
    {
       CalculatedSamplingRate = sampleRate,
       DataChannelCount = apiHandle.GetActiveChannels().Count,
       ChannelBufferSize =  dataSize
    }
}
```

G. DataAnalysisOptions

The options for Data Calculations.

Syntax (C#)

AnalyzeOptions DataAnalyzeOptions

Assemblies:

Lion.Base.dll

Remarks:

This property will give the application the ability to set/get the parameters (options) for processing and calculating the data blocks read from the Channels.

Example:

```
public static void SetOptions(CPL590Api apiHandle, int dataSize, double numberOfRevs)
{
    // Get the System Configuration structure
    var system = apihandle.GetSystem();

    // Set new options
    system.DataAnalyzeOptions = new AnalyzeOptions ()
    {
       NumberRevolutions = numberOfRevs,
       DataChannelCount = apiHandle.GetActiveChannels(),
       ChannelBufferSize =  dataSize
    }
}
```

2.1.2 ChassisConfiguration

ChassisConfiguration	Initializes a new instance of the ChassisConfiguration class.
Add	Adds the Channel to Chassis Configuration.
Get	Gets the specified Channel Configuration
Type	The type of Chassis.
ChassisNumber	The designation number of the Chassis
ChannelList	The list of Channels (Drivers) installed on this Chassis.

The Configuration for each Chassis in the System.

A. ChassisConfiguration

Initializes a new instance of the Chassis**Configuration** class.

Syntax (C#)

void ChassisConfiguration ()

Assemblies:

Lion.Base.dll

Remarks:

This method is automatically called when the GetAvailableChannels method is run on the API and the API has found Devices on the USB Bus. It does not have to be called by the application.

B. Add

Adds the Channel to the ChannelList property.

Syntax (C#)

void Add (ChannelConfiguration item)

Assemblies:

Lion.Base.dll

Remarks:

This function is automatically called when the GetAvailableChannels method is run on the API and the API has found Devices on the USB Bus. On the CPL59X One-Channel System, the will be a different Chassis configuration added for each Device found. It does not have to be called by the application.

C. Get

Get (string SerialNumber)	Gets the specified Channel Configuration by the serial number of the corresponding Channel
Get (int ChannelNumber)	Gets the specified Channel Configuration by the channel number

Syntax (C#)

ChannelConfiguration Get (string SerialNumber)

ChannelConfiguration Get (int ChannelNumber)

Assemblies:

Lion.Base.dll

Remarks:

This method is called when the application needs to retrieve the configuration for one of the Channels installed in the Chassis. The Channel selected will be the one with the corresponding Serial Number or Channel Number. If the Serial Number string is empty, or the Channel Number is -1, then the first Channel in the list will be retrieved.

Example:

```
public static int GetChannelNumber(CPL590Api apiHandle, string serialNumber)
{
    // Get the System Configuration structure
    var system = apihandle.GetSystem();

    // Get the desired Chassis Configuration structure
    var chassis = apihandle.GetChassis();

    // Get the desired Channel Configuration structure
    var channel = chassis?.Get(serialNumber);

     //  Return with the number of this Channel.
     return channel.ChannelNumber;
}
```

D. Type

The type of Channel.

Syntax (C#)

ChassisType Type

Assemblies:

Lion.Base.dll

Remarks:

This property allows the Application to set which type of Channel is created for data transfer. For CPL59X, this type is only set to *Data*.

E. ChassisNumber

The designation number of the Chassis.

Syntax (C#)

int ChassisNumber

Assemblies:

Lion.Base.dll

Remarks:

This property is incremented by one for each new Chassis in the System found by the API. The first Chassis is designated as 1.

F. ChannelList

The strong typed List of all Channels found in this Chassiss.

Syntax (C#)

List<ChannelConifguration> ChassisList

Assemblies:

Lion.Base.dll

Remarks:

This list is filled when the Add function is run during the GetAvailableChannels method on the API and the API has found Devices on the USB Bus. It does not have to be called by the application.

2.1.3 ChannelConfiguration

The Configuration for each Channel (Driver) in the Chassis.

Type	The type of Channel.
ChannelNumber	The designation number of the Channel
IsEnabled	The flag indicating whether the Channel is enabled
IsConnected	The flag indicating whether the Channel is connected
Name	The name of the Channel.
Id	The identifier of the Channel.
Calibration	The Calibration (TEDs) data for the device Channel.
Measurements	The Measurements of the Data received on the Channel.
DriverHandle	The driver handle.

A. Type

The type of Channel.

Syntax (C#)

ChannelType Type

Assemblies:

Lion.Base.dll

Remarks:

This property allows the Application to set which type of Channel is created for data transfer. For CPL59X, this type is only set to *Data*.

B. ChannelNumber

The designation number of the Channel

Syntax (C#)

int ChannelNumber

Assemblies:

Lion.Base.dll

Remarks:

This property is incremented by one for each new Device (Driver) in the Chassis found by the API. The first Channel is designated as 1. If there are multiple Chassis's in the System, this number will correspond to the Slot numbers in each Chassis.

Example:

```
public static int GetChannelNumber(CPL590Api apiHandle, string serialNumber)
{
    // Get the System Configuration structure
    var system = apihandle.GetSystem();

    // Get the desired Chassis Configuration structure
    var chassis = apihandle.GetChassis();

    // Get the desired Channel Configuration structure
    var channel = chassis?.Get(serialNumber);

     //  Return with the number of this Channel.
     return channel.ChannelNumber;
}
```

C. IsEnabled

The flag indicating whether the Channel is enabled for connection.

Syntax (C#)

bool IsEnabled

Assemblies:

Lion.Base.dll

Remarks:

This property is to be set by the Application before Connect is called. The flag informs the API that the Channel is to be opened. This allows the Application to decide which Channel(s) should be run and tested. If the flag is not set, then the API will not open connection to the Driver.

Example:

```
public static void Main()
{
    // Get API
    var apiHandle = CPL590Api();

    // Get List of Devices on USB Bus
    var deviceList = apiHandle.GetAvailableChannels();

    // Cycle through the list and enable all channels
    foreach (var driver in deviceList)
    {
       // Only enable Channel (Driver) #1 for testing
       if (driver.ChannelNumber < 2)
            driver.IsEnabled = true;
    }

    // Connect to Channels (Drivers)
    apiHandle.Connect(deviceList);

    // Cycle through the list and check channels
    foreach (var driver in deviceList)
    {
       // Only check Channel (Driver) #1 for connection
       if (driver.ChannelNumber > 1 )
            continue;

       // Check Connected flag
       if (driver.IsConnected == false)
       {
           Console.Write("Connect Failed.");
       }
    }
}
```

D. IsConnected

The flag indicating whether the Channel has opened for testing.

Syntax (C#)

bool IsConnected

Assemblies:

Lion.Base.dll

Remarks:

This property is to be set by the Application after Connect has been successful.

Example:

```
public static void Main()
{
    // Get API
    var apiHandle = CPL590Api();

    // Get List of Devices on USB Bus
    var deviceList = apiHandle.GetAvailableChannels();

    // Cycle through the list and enable all channels
    foreach (var driver in deviceList)
    {
       // Only enable Channel (Driver) #1 for testing
       if (driver.ChannelNumber < 2)
            driver.IsEnabled = true;
    }

    // Connect to Channels (Drivers)
    apiHandle.Connect(deviceList);

    // Cycle through the list and check channels
    foreach (var driver in deviceList)
    {
       // Only check Channel (Driver) #1 for connection
       if (driver.ChannelNumber > 1 )
            continue;

       // Check Connected flag
       if (driver.IsConnected == false)
       {
           Console.Write("Connect Failed.");
       }
    }
}
```

E. Name

A text string containing the name of this member.

Syntax (C#)

string Name

Assemblies:

Lion.Base.dll

Remarks:

This property was read from the Calibration (TEDs) information stored in the Driver.

Example:

```
public static void Main()
{
    // Get API
    var apiHandle = CPL590Api();

    // Get List of Devices on USB Bus
    var deviceList = apiHandle.GetAvailableChannels();

    // Cycle through the list and display names of Drivers (Channels)
    foreach (var driver in deviceList)
    {
       // Display names of Drivers found
       Console.Write(driver.Name);
    }
}
```

F. Id

The text string identifier of the Channel.

Syntax (C#)

string Id

Assemblies:

Lion.Base.dll

Remarks:

This property is the actual identifier numbers for the ‘A’ and ‘B’ sides of the FTD4222 chip on the CPL59X Drivers. The identifiers for the two sides are separated by a colon (‘;’).

Example:

```
public static void Main()
{
    // Get API
    var apiHandle = CPL590Api();

    // Get List of Devices on USB Bus
    var deviceList = apiHandle.GetAvailableChannels();

    // Cycle through the list and display ID’s of Drivers (Channels)
    foreach (var driver in deviceList)
    {
       // Display names of Drivers found
       Console.Write(driver.Id);
    }
}
```

G. Calibration

The Calibration (TEDs) data for the device Channel.

Syntax (C#)

Calibration Calibration

Assemblies: Lion.Base.dll

Remarks:

This information for this property is read from the TEDs memory of the Driver.

Example:

```
public static void Main()
{
    // Get API
    var apiHandle = CPL590Api();

    // Get List of Devices on USB Bus
    var deviceList = apiHandle.GetAvailableChannels();

    // Cycle through the list and enable all channels
    foreach (var driver in deviceList)
    {
       // Only enable Channel (Driver) #1 for testing
       if (driver.ChannelNumber < 2)
            driver.IsEnabled = true;
    }

    // Connect to Channels (Drivers)
    apiHandle.Connect(deviceList);

    // Cycle through the list and check channels
    foreach (var driver in deviceList)
    {
       // Only check Channel (Driver) #1 for connection
       if (driver.ChannelNumber > 1 )
            continue;

       // Check Connected flag
       if (driver.IsConnected == false)
       {
           Console.Write("Connect Failed.");
       }
    }
}
```

H. Measurements

The Measurement values (Statistics) of the incoming Data.

Syntax (C#)

Measurement Measurements

Assemblies: Lion.Base.dll

Remarks:

The information for this property is calculated on each Data Buffer received.

Example:

```
public static void Main()
{
    // Get API
    var apiHandle = CPL590Api();

    // Get List of Devices on USB Bus
    var deviceList = apiHandle.GetAvailableChannels();

    // Cycle through the list and enable all channels
    foreach (var driver in deviceList)
    {
       // Only enable Channel (Driver) #1 for testing
       if (driver.ChannelNumber < 2)
            driver.IsEnabled = true;
    }

    // Connect to Channels (Drivers)
    apiHandle.Connect(deviceList);

    // Cycle through the list and check channels
    foreach (var driver in deviceList)
    {
       // Only check Channel (Driver) #1 for connection
       if (driver.ChannelNumber > 1 )
            continue;

       // Check Connected flag
       if (driver.IsConnected == false)
       {
           Console.Write("Connect Failed.");
       }
    }
}
```

I. DriverHandle

The pointer to the CPL59X Driver class. This class performs the low-level functions with the actual CPL59X FTD 4222 chip.

Syntax (C#)

bool IsEnabled

Assemblies: Lion.Base.dll

Remarks:

This property is to be set by the Application before Connect is called. The flag informs the API that the Channel is to be opened. This allows the Application to decide which Channel(s) should be run and tested. If the flag is not set, then the API will not open connection to the Driver.

Example:

```
public static void Main()
{
    // Get API
    var apiHandle = CPL590Api();

    // Get List of Devices on USB Bus
    var deviceList = apiHandle.GetAvailableChannels();

    // Cycle through the list and enable all channels
    foreach (var driver in deviceList)
    {
       // Only enable Channel (Driver) #1 for testing
       if (driver.ChannelNumber < 2)
            driver.IsEnabled = true;
    }

    // Connect to Channels (Drivers)
    apiHandle.Connect(deviceList);

    // Cycle through the list and check channels
    foreach (var driver in deviceList)
    {
       // Only check Channel (Driver) #1 for connection
       if (driver.ChannelNumber > 1 )
            continue;

       // Check Connected flag
       if (driver.IsConnected == false)
       {
           Console.Write("Connect Failed.");
       }
    }
}
```

2.1.4 DeviceInfo

Flags	Indicates device state. Can be any combination of the following: FT_FLAGS_OPENED
Type	Indicates the device type. Can be one of the following: FT_DEVICE_232R, FT_DEVICE_2232C, FT_DEVICE_BM, FT_DEVICE_AM, FT_DEVICE_100AX or FT_DEVICE_UNKNOWN
Id	The Vendor ID and Product ID of the device
LocId	The physical location identifier of the device
SerialNumber	The device serial number
Description	The device description
FtHandle	The device handle. This value is not used externally and is provided for information only. If the device is not open, this value is 0.

2.1.5 AsyncState

ChannelCount	The number of Channels
DataSize	The size of the data.
DataBuffer	The data buffer.
Offset	The offset into the buffer (for each channel).
ACallBack	The callback when data is received.

2.1.6 AnalysisOptions

EccentricityChannel	The eccentricity channel. Default is 0.
TypeData	The type of data coming from the device (see DataType for choices) Digital for Count). This flag is used to calculate the proper Displacement values for the type of device. Default is Digital.
SamplingMode	The sampling mode. Default is Manual.
NumberRevolutions	The Number of Revolutions to be used in Auto Sample mode. Default is 20.
SamplesPerRevolution	The Samples per Revolution to be used in Auto Sample mode. Default is 100.
ActualSamplingRate	The actual sampling rate. Default is 4000_{10}.
TargetMultiplier	The target multiplier. Default is 1.0.
UnitScale	The unit scale. Default is Micrometers.
RemoveNG	Indicating whether to remove Near Gap measurement in calculating displacement from raw probe voltage/count. Default is true.
PolarCalculation	Indicating whether to calculate displacement, from raw probe voltage/count, for polar view displays. Default is false.

2.1.7 TransferOptions

DataChannelCount	The Number of Data Channels to be run. Default is 0.
TotalChannelCount	The Total Number of Channels (Data Channels plus Index/Temperature if enabled). Default is 0.
ChannelBufferSize	The receive buffer size (per channel). Default is 2048_{10}.
ReceiveBufferSize	The total receive buffer size (all channels). Default is 2048_{10}.
MaximumRpm	The maximum RPM. Default is 15000_{10}.
CalculatedSamplingRate	The calculated sampling rate. Default is 10000_{10}.
SensitivityRange	The sensitivity range. Default is R1.

2.1.8 Calibration

The calibration (TEDs) information for CPL591/2

Property Type:

Channel	The channel number of this Driver.
Model	The model name of this Driver .
SerialNumber	The serial number of this Driver.
ProbeModel	The probe model name.
ProbeSerialNumber	The probe serial number.
OrderId	The customer order number.
PartNumber	The part number of this Driver.
FarGap	The maximum range (in displacement) calibrated for this Driver.
NearGap	The minimum range (in displacement) calibrated for this Driver.
MaxVolts	The maximum volts calibrated for this Driver.
MinVolts	The minimum volts calibrated for this Driver.
SensitivityVolt	The Analog Sensitivity (Volts per unit) calibrated for this Driver
FarCount	The maximum digital count calibrated for this Driver.
NearCount	The minimum digital count calibrated for this Driver.
Range	The difference between the Far Gap and Near Gap.
SensitivityCount	The Digital Sensitivity (Counts per unit) calibrated for this Driver.
Units	The units for displacement (micrometers, millinches, etc.) calibrated for this Driver

2.1.9 Measurement

The Measurement values (Statistics) of the incoming Data.

Min	The minimum value in the data buffer.
Max	The maximum value in the data buffer.
P2P	The range of values (peak to peak) in the data buffer.
Average	The average value in the data buffer.
Count	The size of the data buffer.

2.2 Enumerations

2.2.1 ApiState

Idle	Idle (initial state)
Connecting	API is connecting to the device (Connect)
Disconnecting	API in Disconnecting from Device
ReadConfigurationSlot1	API is reading Calibration info (TEDs) from driver in slot1
ReadConfigurationSlot2	API is reading Calibration info (TEDs) from driver in slot2
ReadConfigurationSlot3	API is reading Calibration info (TEDs) from driver in slot3
ReadConfigurationSlot4	API is reading Calibration info (TEDs) from driver in slot4
ReadConfigurationSlot5	API is reading Calibration info (TEDs) from driver in slot5
ReadConfigurationSlot6	API is reading Calibration info (TEDs) from driver in slot6
ReadConfigurationSlot7	API is reading Calibration info (TEDs) from driver in slot7
ReadConfigurationSlot8	API is reading Calibration info (TEDs) from driver in slot8
Flush	LionStream is executing a Flush function
Ready	API is ready for next function call
Sampling	LionStream is running Auto Sampling mode and changing Sample rate to get desired Number of Revolutions and Samples per Revolution.
Streaming	LionStream is receiving data from Device
Error	API has detected an error
Shutdown	API is going through the Shutdown sequence
Closed	API is going through the closing sequence

2.2.2 ApiType

Device	The API is for a real Device
Simulator	The API is for a Simulated Device

2.2.3 DeviceMode

Unknown	The unknown
Elite	Elite System
Sca	SpindleCheck Analyzer
Sci	SpindleCheck Inspector
Cpl590	CPL59X 2U

2.2.4 ChannelType

Data	The Channel is a Data Channel
Encoder	The Channel is an Encoder Channel
Index	The Channel is an Index Channel
Temperature	The Channel is an Temperature Channel

2.2.5 ChassisType

Cpl590	CPL59x
Elite	Elite with NI DAQ Device (USB-6366, etc.)
Sci	Spindle Check Device

2.2.6 DataType

Analog	The data coming from the device is in Volts. This flag is used to calculate the proper Displacement values for the type of device.
Digital	The data coming from the device is in Digital Counts. This flag is used to calculate the proper Displacement values for the type of device.

2.2.7 SensitivityLevel

R1	In a single Range Driver (CPL591) this is the only Gap Range selection setting. In a multiple range driver (CPL592 or CPL594), this is the first Gap Range selection setting. The Gap ranges are determined by the Calibration of this Driver.
R2	In a multiple range driver (CPL592 or CPL594), this is the second Gap Range selection setting. The Gap ranges are determined by the Calibration of this Driver.
R3	In a multiple range driver (CPL594), this is the third Gap Range selection setting. The Gap ranges are determined by the Calibration of this Driver.
R4	In a multiple range driver (CPL594), this is the fourth Gap Range selection setting. The Gap ranges are determined by the Calibration of this Driver.

2.2.8 SamplingMode

Auto	Automatic Sampling Mode. API will change Device's sampling rate until the desired number of Revolutions and Samples per Revolution are received from the device in a data block.
Manual	Manual Sampling Mode. The Device's sampling rate can be manually set to any value (above 0) with no regard to number of Revolutions or Samples per Revolution.
Encoder	Encoder Sampling Mode. This mode is only for the Elite Systems with a TMP190 board connected to a Spindle that has an Encoder output.

2.2.9 TransferOptionName

CalculatedSamplingRate	The calculated sampling rate. This is used for Auto Sampling mode.
ChannelBufferSize	The channel buffer size. The amount of data that each Channel will read from the Device (Driver) in 16 bit words.
ReceiveBufferSize	The channel buffer size. The total amount of data (all Channels) that will be read from the Device (Driver) in 16 bit words.
SensitivityRange	The sensitivity range setting for the Driver.
DataChannelCount	The data channel count. The number of Driver Channels that are running Data transfer, excluding Index and Temperature Channels. Index and Temperature Channels are not used in current CPL59X Systems.
TotalChannelCount	The total channel count. The number of Driver Channels that are running including Index and Temperature Channels. Index and Temperature Channels are not used in current CPL59X Systems.
All	All of the Transfer Options

2.2.10 AnalysisOptionName

EccentricityChannel	The eccentricity channel. Default is 0.
SamplingMode	The sampling mode. Default is Manual.
NumberRevolutions	The Number of Revolutions to be used in Auto Sample mode. Default is 20.
SamplesPerRevolution	The Samples per Revolution to be used in Auto Sample mode. Default is 100.
ActualSamplingRate	The actual sampling rate. Default is 4000_{10}.
TargetMultiplier	The target multiplier. Default is 1.0.
UnitScale	The unit scale. Default is Micrometers.
RemoveNG	Indicating whether to remove Near Gap measurement in calculating displacement from raw probe voltage/count. Default is true.
PolarCalculation	Indicating whether to calculate displacement, from raw probe voltage/count, for polar view displays. Default is false.
All	All of AnalysisOptions

3.0 Lion.Stream

3.1 Public Member Functions

BeginRead	Begins an asynchronous read operation.
CopyTo	Reads the double words from this stream and writes them to generic class Stream in bytes, using a specified buffer size.
CopyToAsync	Reads the double words from this stream and writes them to generic class Stream in bytes, using a specified buffer size in asynchronous mode.
EndRead	Waits for the pending asynchronous read operation to complete.
Flush	Clears buffers for this stream and causes any buffered data to be disposed.
Read	Reads data from the Device
ReadAsync	Asynchronously Reads data from the Device
Dispose	Releases the managed and unmanaged resources used by API

3.1.1 BeginRead

Begins an asynchronous read operation. (Consider using ReadAsync instead.)

Syntax (C#)

IAsyncResult BeginRead (double *array*[,], int *numBytes*, AsyncCallback *userCallback*, object *stateObject*)

Assemblies: Lion.Core.dll

Parameters:

array	The buffer to read data into.
offset	The byte offset in array at which to begin reading.
numBytes	The maximum number of double words to read.
userCallback	The method to be called when the asynchronous read operation is completed.
stateObject	A user-provided object that distinguishes this particular asynchronous read request from other requests. Refer to AsyncState property for the structure.

Returns:

An IAsyncResult that represents the asynchronous call.

Remarks:

The BeginRead method starts asynchronously reading data from the CPL591/2 Driver. Calling the BeginRead method gives you the ability to receive data within a separate execution thread.
You must create a callback method that implements the *AsyncCallback* delegate and pass its name to the BeginRead method. Because you will want to obtain the received data within your callback method, you should create a small class or structure to hold a read buffer and any other useful information. Pass the structure or class instance to the BeginRead method through the *state* parameter.
Your callback method should call the EndRead method. When your application calls BeginRead, the system will wait until data is received or an error occurs, and then the system will use a separate thread to execute the specified callback method, and blocks on EndRead until the provided LionStream reads data or throws an exception. If you want the original thread to block after you call the BeginRead method, use the WaitOne method. Call Set in the callback method when you want the original thread to continue executing. For additional information about writing callback methods, see Marshaling a Delegate as a Callback Method.

The BeginRead method reads as much data as is available, up to the number of bytes specified by the *size* parameter

Exceptions:

InvalidOperationException	**Device** must be Connected.
ArgumentException	Encoder Mode not supported by this Device *or* Missing Callback *or* Sampling Rate must be at least 10. *or* Data size must be at least 100 words.

Example:

```
LionStream lionStream;
double[] testData ;

public static public static void DataHandler(CPL590Api apiHandle, int dataSize)
{
    // Get the number of Channels
    var channelCount = apihandle.GetOption(TransferOptionName. DataChannelCount);

    // Set the data size
    apihandle.SetOption(TransferOptionName.ChannelBufferSize, dataSize);

    // Build test data buffer
    testData = new double[channelCount, dataSize];

     //  Use the GetStream public method to return the LionStream.
     lionStream = apiHandle.GetStream();

     // Perform a BeginRead function to start receiving data
     lionStream.BeginRead(testData, 0, dataSize, DataCallbackS, null);

     //  If Begin did not happen, error
     if (!apiHandle.Running)
    {
        Console.Write("Data Did NOT Begin!")
    }
}

public void DataCallbackS(IAsyncResult ar)
{
    //Get the data packets from the Api
     var read = lionStream.EndRead(ar);
    {
         var stateObject =  (AsyncState)ar.AsyncState;
         testData = stateObject.DataBuffer;
    }
}
```

3.1.2 CopyTo

Reads the double words from this stream and writes them to generic class Stream in bytes, using a specified buffer size.

Syntax (C#):

double[,] CopyTo(Stream destination, int channelNumber, int bufferSize)

Assemblies: Lion.Core.dll

Parameters:

destination	The stream to which the contents of the current stream will be copied.
channelNumber	The channel number. The default is 0
bufferSize	The size of the buffer (in double words). This value must be greater than zero. The default size is 2048.

Returns:

The data buffer (in 2D array of double words) received from the CPL591/2 Driver

Exceptions:

ArgumentException	Invalid Channel Number. Check number of channels created.

Remarks:

Example:

```
LionStream lionStream;
double[] testData ;

public static void CopyData(CPL590Api apiHandle, MemoryStream objMemoryStream, int dataSize)
{
    // Get the number of Channels
    var channelCount = apihandle.GetOption(TransferOptionName. DataChannelCount);

    // Set the data size
    apihandle.SetOption(TransferOptionName.ChannelBufferSize, dataSize);

    // Build test data buffer
    testData = new double[channelCount, dataSize];
    for (var ptr = 0; ptr < dataSize; ptr++) testData[ptr] = 0x69;

     //  Use the GetStream public method to return the LionStream.
     lionStream = apiHandle.GetStream();

    // Copy LionStream to Memory Stream using CopyTo method
    var testDataAll = lionStream.CopyTo(objMemoryStream, 0, dataSize);

    objMemoryStream.Seek(0, SeekOrigin.Begin);

    // Build check data buffer
    var checkBytes = new byte[dataSize * 2];
    for (var ptr = 0; ptr < 100; ptr++) checkBytes[ptr] = 0x5A;

    // Read data back from Memory Stream
    var byteCount = objMemoryStream.Read(checkBytes, 0, checkBytes.Length);
    checkData = checkBytes.ByteToDouble();

    // Verify copied data is correct
    var testData = testDataAll.CopyRow(0);
    for (var index = 0; index < dataSize; index++)
    {
         if (checkData[index] != testData[index])
         {
             Console.Write($"Stream Data Compare Error at Index {index}");
             return ;
         }
    }
}
```

3.1.3 CopyToAsync

Asynchronously reads the double words from this stream and writes them to generic class Stream in bytes, using a specified buffer size.

Syntax (C#):

Task<double[,]> CopyToAsync(Stream destination, int channelNumber, int bufferSize, CancellationToken cancellationToken)

Assemblies: Lion.Core.dll

Parameters:

destination	The stream to which the contents of the current stream will be copied.
channelNumber	The channel number. The default is 0
bufferSize	The size of the buffer (in double words). This value must be greater than zero. The default size is 2048.

Returns:

The data buffer (in 2D array of double words) received from the CPL591/2 Driver

Exceptions:

ArgumentException	Invalid Channel Number. Check number of channels created.

Remarks:

Example:

```
LionStream lionStream;
double[] testData ;

public static void CopyData(CPL590Api apiHandle, MemoryStream objMemoryStream, int dataSize)
{
    // Get the number of Channels
    var channelCount = apihandle.GetOption(TransferOptionName. DataChannelCount);

    // Set the data size
    apihandle.SetOption(TransferOptionName.ChannelBufferSize, dataSize);

    // Build test data buffer
    testData = new double[channelCount, dataSize];
    for (var ptr = 0; ptr < dataSize; ptr++) testData[ptr] = 0x69;

     //  Use the GetStream public method to return the LionStream.
     lionStream = apiHandle.GetStream();

    // Copy LionStream to Memory Stream using CopyTo method
    var testDataAll = lionStream.CopyToAsync(objMemoryStream, 0, dataSize,
                                                          taskCancel.Token).Result;

    objMemoryStream.Seek(0, SeekOrigin.Begin);

    // Build check data buffer
    var checkBytes = new byte[dataSize * 2];
    for (var ptr = 0; ptr < 100; ptr++) checkBytes[ptr] = 0x5A;

    // Read data back from Memory Stream
    var byteCount = objMemoryStream.Read(checkBytes, 0, checkBytes.Length);
    var checkData = checkBytes.ByteToDouble();

    // Verify copied data is correct
    var testData = testDataAll.CopyRow(0);
    for (var index = 0; index < dataSize; index++)
    {
         if (checkData[index] != testData[index])
         {
             Console.Write($"Stream Data Compare Error at Index {index}");
             return ;
         }
    }
}
```

3.1.4 EndRead

Waits for the pending asynchronous read operation to complete.

Syntax (C#):

```
int EndRead(IAsyncResult asyncResult)
```

Assemblies: Lion.Core.dll

Parameters:

asyncResult	The reference to the pending asynchronous request to wait for.

Returns:

The number of double words read from the stream, between 0 and the number of double words you requested. Streams only return 0 at the end of the stream, otherwise, they should block until at least 1 byte is available.

Exceptions:

Remarks:

In the .NET Framework 4 and earlier versions, you have to use methods such as BeginRead and EndRead to implement asynchronous I/O operations. These methods are still available in the .NET Framework 4.5 to support legacy code; however, the new async methods, such as ReadAsync, CopyToAsync, and FlushAsync, help you implement asynchronous I/O operations more easily.

Call EndRead to determine how many bytes were read from the stream.

EndRead can be called once on every IAsyncResult from BeginRead.

This method blocks until the I/O operation has completed.

Example:

(see the BeginRead function for example)

3.1.5 Flush

When overridden in a derived class, clears all buffers for this stream and causes any buffered data to be written to the underlying device.

Syntax (C#):

```
public void Flush ();
```

Assemblies: Lion.Core.dll

Remarks:

Override Flush on streams that implement a buffer. Use this method to move any information from an underlying buffer to its destination, clear the buffer, or both. Depending upon the state of the object, you might have to modify the current position within the stream (for example, if the underlying stream supports seeking).

Exceptions:

IO.IOException	An I/O error occurred.
ObjectDisposedException	The stream is closed.

3.1.6 Read

Reads a block of double words from the stream and writes the data in a given buffer.

Syntax (C#):

public int Read (double array[,], int count)

Assemblies: Lion.Core.dll

Parameters:

array	When this method returns, contains the specified data array with the values between *offset* and (*offset* + *count* - 1*)* replaced by the double words read from the current source.
offset	The byte offset in *array* at which the read double words will be placed.
count	The maximum number of double words to read.

Returns:

The total number of double words read into the buffer. This might be less than the number of double words requested if that number of double words are not currently available, or zero if the end of the stream is reached.

Exceptions:

InvalidOperationException	– Device must be Connected. *or* There must be at least one Channel created.
ArgumentException	– Encoder Mode not supported by this Device. *or* Missing Callback. — Sampling Rate must be at least 10. – *or* Data size must be at least 100 words.

Remarks:

Use the CanRead property to determine whether the current instance supports reading. Use the ReadAsync method to read asynchronously from the current stream.

3.1.7 ReadAsync

Reads a block of double words from the stream and writes the data in a given buffer.

Syntax (C#):

public int Read (double array[,], int count)

Assemblies: Lion.Core.dll

Parameters:

array	When this method returns, contains the specified data array with the values between *offset* and (*offset* + *count* - 1*)* replaced by the double words read from the current source.
offset	The byte offset in *array* at which the read double words will be placed.
count	The maximum number of double words to read.

Returns:

The total number of double words read into the buffer. This might be less than the number of double words requested if that number of double words are not currently available, or zero if the end of the stream is reached.

Exceptions:

InvalidOperationException	– Device must be Connected. *or* There must be at least one Channel created.
ArgumentException	– Encoder Mode not supported by this Device. *or* Missing Callback. — – Sampling Rate must be at least 10. *or* Data size must be at least 100 words.

Remarks:

Use the CanRead property to determine whether the current instance supports reading. Use the ReadAsync method to read asynchronously from the current stream.

3.1.8 Dispose

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

Syntax (C#):

public void Dispose ()

Assemblies: Lion.Core.dll

4.0 CPL590 Data Receiving Examples

4.1 Manual Mode

This code example performs a Read from the Driver in Manual mode.

```
public static Main()
{
     // Get API
     var apiHandle = CPL590Api();

     // Get List of Devices on USB Bus
     var deviceList = apiHandle.GetAvailableChannels();

     // Cycle through the list and enable all channels
     foreach (var driver in deviceList)
     {
         driver.IsEnabled = true;
     }

     // Connect to Devices and enable them
     apiHandle.Connect(deviceList);

     double[,] cpl590Data = OpenRead(apiHandle, 5000, 67000, SamplingMode.Manual);

     (put code here to analyze or display data)
}

public static double[,] OpenRead(CPL590Api apiHandle, int dataSize, int sampleRate, SamplingMode mode)
{
     // Get Configuration (TEDs) data from the Driver
     var tedsData = apiHandle.GetCalibrationsAsync(ct).Result;

     // Set the Sampling Rate
     apihandle.SetOption(TransferOptionName.CalculatedSamplingRate, sampleRate);

     // Set the Sensitivity
     apihandle.SetOption(TransferOptionName.SensitivityRange, SensitivityLevel.R2);

     // Set the Target Multiplier
     apihandle.SetOption(AnalysisOpttionName.TargetMultiplier, 0.975);

     // Set the Sampling Mode
     apihandle.SetOption(AnalysisOpttionName.SamplingMode, mode);

     // Set the Data Size
     apihandle.SetOption(TransferOptionName.ChannelBufferSize, dataSize);

     // Get Actives List
     List<ChannelConfiguration> activeList = apiHandle.GetActiveChannels();

     // Build test data buffer
    var testData = new double[activeList.Count, dataSize];

    // Use the GetStream public method to return the LionStream.
     var lionStream = apiHandle.GetStream();

     // Begin Data Transfer from Driver
     var wordsread = lionStream.ReadAsync(testData, dataSize, ct).Result;
     if (wordsread < dataSize)
     {
         Console.Write($"Data Count {wordsread} is less than expected {dataSize}!"); return null;
     }
     return testData;
}
```

4.2 Auto Mode

This code example performs a Read from the Driver in Auto mode. The operator selects the desired number of revolutions and samples per revolution to be present in the data buffer.

```
public static Main()
{
     // Get API
     var apiHandle = CPL590Api();

     // Get List of Devices on USB Bus
     var deviceList = apiHandle.GetAvailableChannels();

     // Cycle through the list and enable all channels
     foreach (var driver in deviceList)
     {
         driver.IsEnabled = true;
     }

     // Connect to Devices and enable them
     apiHandle.Connect(deviceList);

     double[,] cpl590Data = OpenRead(apiHandle, 5000, 20, 100, SamplingMode.Auto);

     (put code here to analyze or display data)
}

public static double[,] OpenRead(CPL590Api apiHandle, int dataSize, , int numberRevs, int samplesPerRev,
                                                                                SamplingMode mode)
{
     // Get Configuration (TEDs) data from the Driver
     var tedsData = apiHandle.GetCalibrationsAsync(ct).Result;

     // Set the Sampling Rate
     apihandle.SetOption(TransferOptionName.CalculatedSamplingRate, 4000);

     // Set the Number Revolutions
     apihandle.SetOption(AnalysisOpttionName.NumberRevolutions, numberRevs);

     // Set the Samples Per Revolution
     apihandle.SetOption(AnalysisOpttionName.SamplesPerRevolution, samplesPerRev);

     // Set the Sensitivity
     apihandle.SetOption(TransferOptionName.SensitivityRange, SensitivityLevel.R1);

     // Set the Target Multiplier
     apihandle.SetOption(AnalysisOpttionName.TargetMultiplier, 0.975);

     // Set the Sampling Mode
     apihandle.SetOption(AnalysisOpttionName.SamplingMode, mode);

     // Set the Data Size
     apihandle.SetOption(TransferOptionName.ChannelBufferSize, dataSize);

     // Get Actives List
     List<ChannelConfiguration> activeList = apiHandle.GetActiveChannels();

     // Build test data buffer
     var testData = new double[activeList.Count, dataSize];

     // Use the GetStream public method to return the
     LionStream. var lionStream = apiHandle.GetStream();

     // Begin Data Transfer from Driver
     var wordsread = lionStream.ReadAsync(testData, dataSize, ct).Result;
     if (wordsread < dataSize)
     {
         Console.Write($"Data Count {wordsread} is less than expected {dataSize}!"); return null;
     }
     return testData;
}
```