Series 835

# Granville-Phillips® Series 835 Vacuum Quality Monitor™ Programmer's Reference Manual


## Programmer's Reference Manual

Reference Manual part number 835003

Revision F – July 2015


## *Customer Service / Technical Support*

**MKS, Granville-Phillips Division**
6450 Dry Creek Parkway
Longmont, CO 80503 USA

Phone: 1-303-652-4400 or 1-800-776-6543
FAX: 1-303-652-2844
Email: *gp-csr@mksinst.com*

**Corporate Office**
**MKS Instruments, Inc.**
2 Tech Drive, Suite 201
Andover, MA 01810 USA

Phone: 1-978-645-5500
www.mksinst.com

# Table of Contents

# 1 Overview

This Programmer's Reference Manual describes both the .NET Application Programming Interface (API) and LabVIEW Virtual Instruments (VIs) available to programmers that want to develop their own applications to communicate with the 835 VQM Controller.

Section 3 (CServiceWrapper Methods (.NET interface)) describes the API calls for use with .NET development. Examples of its use are provided in the Section 5.

Section 4 (LabVIEW VIs) describes the VIs to use for LabVIEW 2011 development.

Section 5 has .NET Programming examples.

Section 8 includes Troubleshooting assistance.

## 1.1 Installation and Quick Start Instructions

The LabVIEW VIs are LabVIEW 2011 VIs. If you will be using LabVIEW for development, completely finish the LabVIEW 2011 install and reboot before attempting the service install. Most of the VIs are thin wrappers for the .NET APIs delivered along with the Viewer Application. The Viewer Application uses the same calls that you will be using when you develop your application. If the Viewer Application is working and able to connect to the controller(s), you know that you have a functioning .NET API installation.

You need to be an administrator to successfully install LabVIEW, the API, associated Service, and Viewer Application. By default, the installation of the API places the associated files into <install directory>\API files[1]. When you run an application (including the Viewer Application) that talks to the Controller, a Windows Service is started that performs the actual serial communications. The Service ("835 Series Service" in the Windows Service dialog) should automatically start when the application runs (or may already be running if previously started). If it does not start automatically you can manually start the Services by opening Control Panel -> Administrative Tools -> Services. The names of the Services are "835 Series ControllerN", where N is a number between 1 and 4. If you exit from your application without calling CloseConnection, you will need to run 800SeriesServiceRestart.cmd, a tool that is provided with this installation in the <install directory>[1]\Utilities directory. Either method for starting/restarting services requires administrator privileges on your system.

In Windows 7, run your program with administrative privileges if the UAC is turned on. To do this, right-click on the executable (not a shortcut) and select "Run as Administrator". Note that the same is true if you are running the Viewer Application; right-click on the executable in the install directory.

If you are developing in Visual Studio and running the debugger, you may need to set up Visual Studio to always run as an administrator. To do this, see the example given at:
http://www.a2zmenu.com/Miscellaneous/Visual-Studio-Always-Run-As-Administrator.aspx

Only one connection to a controller is a Master, which means that only one application can set the setpoints and behavior (for example, start/stop scanning or changing NVRAM values) of the Controller. If the Viewer Application is running, it is Master. Therefore, if you want your application to run as a Master, you need to shut down the Viewer application before you try to connect to the controller.

If you are developing a .NET application, you will need to add a reference to the dlls CServiceWrapper.dll and Utilitites.dll. These are in the <install directory>\API files\NET.

If unexpected errors or events occur, it may be necessary to unplug the USB cable, cycle power on the Controller, re-plug the USB cable, and/or restart the Service to recover. Occasionally, if a USB port is hung or unresponsive, you may need to use a different USB port or restart your computer. Note that the 835 VQM controller is a USB 2.0 device. The controller may not be fully compatible with some computer manufacturer

[1] <install directory>: C:\Program Files (x86)\GP\VQM or C:\Program Files\GP\VQM

implementations of USB 3.0. Refer to the troubleshooting section of the instruction manual if having connection problems on a USB 3.0 port.

See Section 2.1 Normal Operation for more information about the API use. Section 4 gives details on the support for LabVIEW VI use.

## 1.2 Special Considerations for Remote Connections

GetValidConnections has a parameter of type EServiceConnectionType that allows for a connection to a remote machine. The default is to get valid connections for the local machine. To use the API on a machine that is different than the one connected to the controller, set the EServiceConnectionType to TCP and set the last parameter (a string) to the machine's name or IP address. The remote machine must be accessible via the network for this to work. For example:

```
VQM_800_ERROR GetValidConnections(out CDeviceConnectionInfo[] connectionInfoList,
        CDeviceConnectionInfo.EServiceConnectionType serviceConnectionType =
        CDeviceConnectionInfo.EServiceConnectionType.TCP,
        string serviceConnection = "machineID")
```

When debugging your custom application which connects to a controller remotely, remember that there are timeouts that come into play. For example, if you don't communicate with the remote Service for the timeout period, it will disconnect you. The result is that you may find yourself establishing a connection and then losing it unexpectedly.

You may want to check the Service Log for the connection on the remote machine to help debug failures. There is one Service Log for each of the installed Services. For example, if you are only running one connection, the service log you are interested in is located in the <public documents>[2] directory with the file name "835 Series Controller1_error.log".

[2] <public documents>: C:\Users\Public\Documents\VQM\ErrorLogs (Windows 7) or C:\Documents and Settings\All Users\Documents\VQM\ErrorLogs (Windows XP)

# 2 API Class Definition


**Figure 1: API Class Public Methods Overview**

## 2.1 Normal Operation

To communicate with the 835 Series Controller using the API, a common series of commands needs to be sent to the Controller in a specific order. (See Figure 2: Normal Flow)

1. GetValidConnections – This command will return a list of CDeviceConnectionInfo objects that can be used to directly connect to the Controller of your choice. It is possible to set up a CDeviceConnectionInfo manually if you know the information beforehand, but this is not recommended.
2. ConnectToDevice – This command uses the CDeviceConnectionInfo object selected from the list returned from GetValidConnections and attempts to connect to the Controller.
3. If a successful connection is made, the commands listed in this API document can be used to get/set values, get data, etc., from the Controller.
4. DisconnectFromDevice – Unless your application is closing, you will want to call disconnect when you are done getting information from the Controller. This will terminate the selected connection.
5. CloseConnection – This command terminates all connections to all Controllers that your application is connected to and will do some cleanup of various things running inside the API. This command needs to be called before exiting your program and may cause an exception to occur if you do not call it before exiting your application. For example, you may want to include the following code:

```
private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
        api.CloseConnection();
}
```

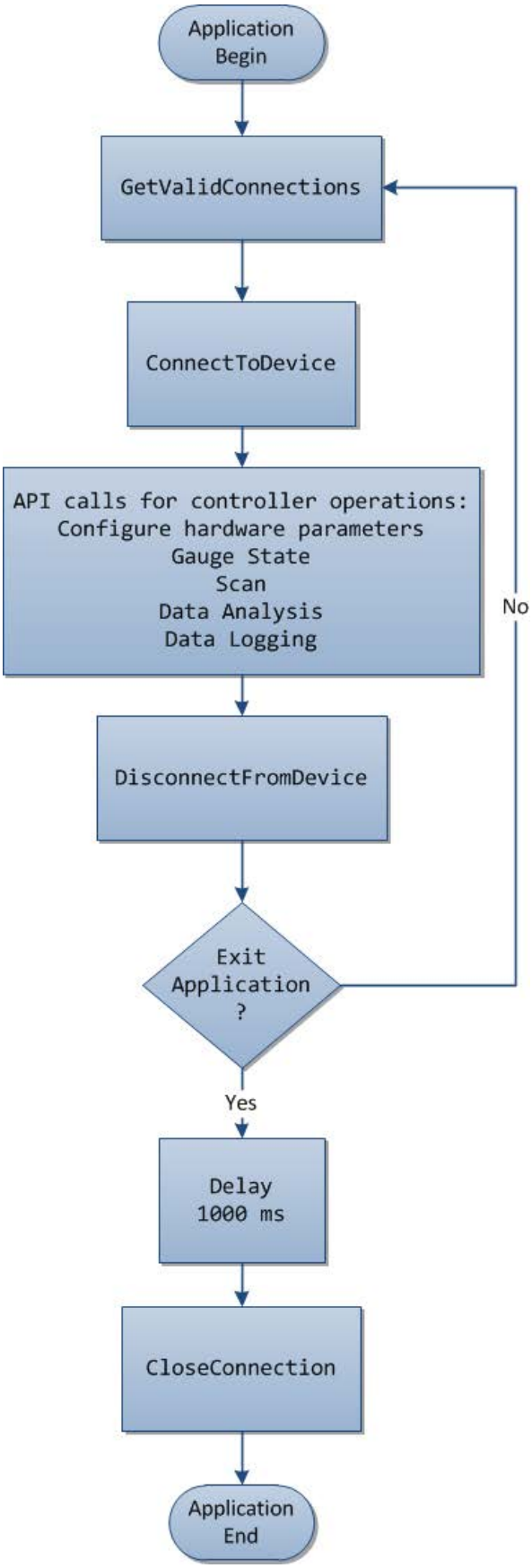



**Figure 2: Normal Flow**

## 2.2 API Utility Classes, Structures, Enumerations

### 2.2.1 VQM_800_ERROR

There are multiple sources of errors. Those generated by the Services as a result of inappropriate requests or internal errors, which are returned via the VQM_800_ERROR.

**VQM_800_ERROR** is an encapsulation of the three parts used for error handling:

- **EErrorType** is an enumeration that starts and ends in the custom error code ranges (5000 through 9999).
- **ErrorString**is the string either returned by the Controller or, if no such string exists, the string used in the user manual.
- **ErrorTroubleShooting** is a detailed string that suggests corrective actions for the user.

An example of getting the error message and troubleshooting tips from the VQM_800_ERROR returned by the service API call is provided in Section 5 of this document.

### 2.2.2 Reported Errors

There are three sources for 835 errors: the software on the PC, the firmware on the controller, and the firmware on the 390802 TPMK. Other errors may be generated by Windows or .NET.

**EErrorType**

EErrorType is an enumeration used for the error type field of VQM_800_ERROR. See EErrorType in 2.2.1 VQM_800_ERROR for restrictions on the enumeration. “NO_ERROR” has a reserved value of 0 for compatibility with LabVIEW. It has the following values:

| Name | Value |
|---|---|
| NO_ERROR | 0 |
| NET_EXCEPTION | 5001 |
| NO_CONTROLLERS_FOUND | 5002 |
| COMMUNICATION_EXCEPTION | 5003 |
| INSUFFICIENT_PRIVELEGES | 5004 |
| NO_HEADER_DATA | 5005 |
| NO_CONNECTION_TO_SERVICE | 5006 |
| FAILED_TO_CONNECT_TO_CONTROLLER | 5007 |
| NO_MORE_AVAILABLE_SERVICES | 5008 |
| NO_SERVICES_INSTALLED | 5009 |
| SERVICE_FAILED_TO_START | 5010 |
| CANNOT_CONNECT_TO_SERVICE | 5011 |
| UNKNOWN_SERVICE_CONNECTION_ERROR | 5012 |
| INVALID_PATH | 5013 |
| CONTROLLER_ERRORS | 5014 |
| NO_ANALYZED_DATA | 5015 |
| UNKNOWN_ERROR | 5016 |
| LABVIEW_EXCEPTION | 5017 |
| INVALID_CONNECTION_DEVICE | 5018 |
| INVALID_SCAN_COUNT | 5019 |
| SCAN_TIMEOUT | 5020 |
| COMBOFILE_DATA_UNAVAILABLE | 5021 |
| UNABLE_TO_AUTO_TUNE | 5022 |
| AUTO_TUNE_BUSY | 5023 |
| SERVICE_INITIALIZATION_ERROR | 5024 |
| CONTROLLER_REMOVED_UNEXPECTEDLY | 5025 |
| START_AUTO_TUNE_ERROR | 5026 |
| CONTROLLER_NOT_CONNECTED_ERROR | 5027 |
| SCPI_COMMAND_FAILED_ERROR | 5028 |
| SET_CONTROLLER_STANDBY_FAILED | 5029 |
| SUBOPTIMAL_AUTO_TUNE_RESULTS | 5030 |
| SATURATION_RISK | 5031 |
| INVALID_SETPOINT_PARAMETER | 5032 |
| INVALID_GAUGE_STATE | 5033 |
| AUTO_TUNE_NOISE_TOO_HIGH | 5034 |
| AUTO_TUNE_CONTROLLER_ERROR | 5035 |
| EM_GAIN_LIMIT_REACHED | 5036 |

**Figure 3: EErrorType enumerations**

### Controller Errors

Errors generated by the Controller (type 5014) are detailed via calls to GetErrorCountAndQueue. To assure that each application with a connection has an opportunity to see these errors, the Service stores a list of errors for each open connection. (The error count is the number of entries in the queue.) The Controller itself supports only 20 errors in the error queue and reports an overflow after that.

When the Error Queue is read for a particular connection, its queue contents are reset.

#### 390802 Errors

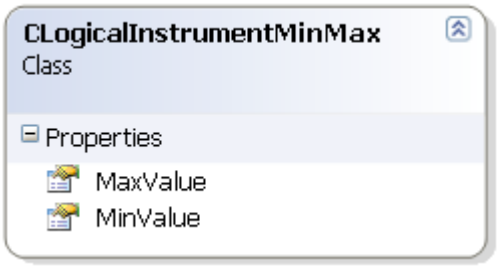
The TPMK (390802), if used, may generate errors also. These error conditions are returned in the Header information. The user is responsible for monitoring the 390802 status and for performing associated behaviors. The Service monitors the 390 status as well to assure that the Total Pressure is valid.

#### Windows and .NET Errors

Errors generated by Windows or .NET (type 5001) are further specified by the ErrorString and ErrorTroubleShooting strings.

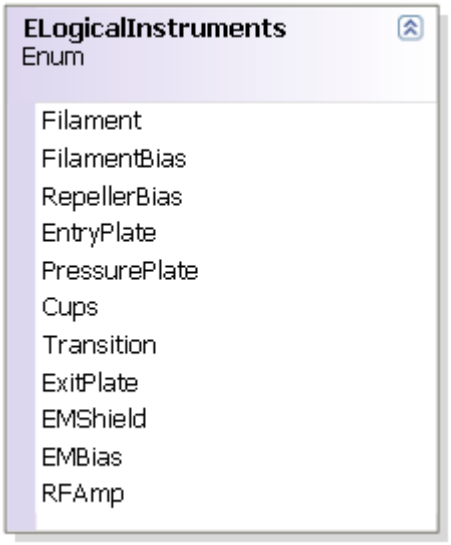
### 2.2.3 CLogicalInstrumentMinMax

CLogicalDriveMinMaxSetpoint defines the valid range for a setpoint via a minimum and maximum value (read from the Controller).



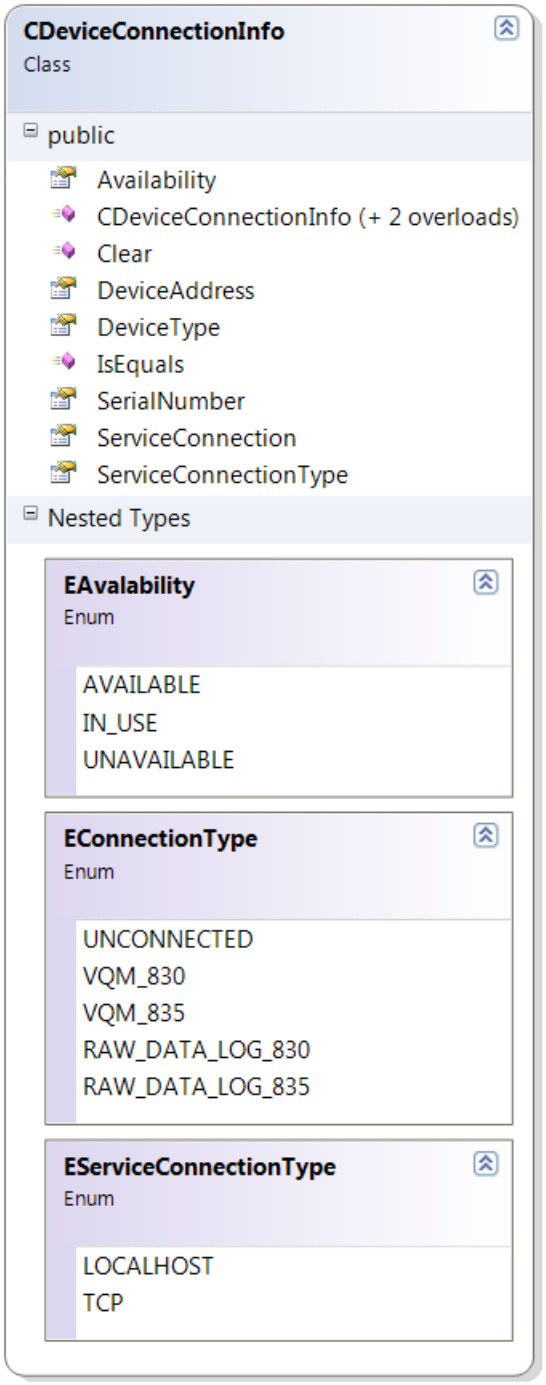

### 2.2.4 ELogicalInstruments

The ELogicalInstruments enumeration identifies the values to use when writing to a particular logical instrument on the Controller.



### 2.2.5 CDeviceConnectionInfo

The user will set the DeviceAddress and DeviceType based on the values returned from the GetValidConnections call. The class will then be used to open a connection with the Controller via ConnectToDevice. The same CDeviceConnectionInfo should be used to interact with the Controller. Multiple instances can be used to talk to more than one Controller.




The DeviceAddress should be set to the COM port name, for example "COM8".

The DeviceType should be set to the device's type. For example, "EConnectionType.VQM_835".

SerialNumber is a string representation of the controller's serial number which can be used to distinguish it from other Controllers.

EAvailability is the current availability of the Controller.

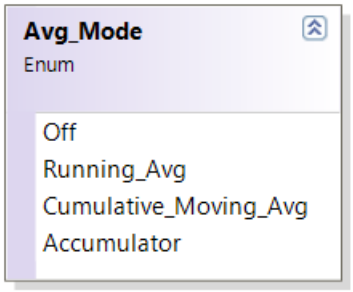
- **AVAILABLE** – Can connect to the Controller and there is no current connection to the chosen Controller, so the Controller will connect as a “master”.
- **IN_USE** – Can connect to the Controller, but there is already a connection to the specified Controller, resulting in a “monitor” connection.
- **UNAVAILABLE** – The specified Controller is not present, or is in use by an application that is not using the API to connect. (ie Hyper-term, Termite, etc). This status will be returned when Windows can not access the USB port. You may need to unplug/plug the USB cable, restart the controller, or restart the Service to recover from this situation.

### 2.2.6 Data Analysis Classes

Programming examples for data analysis are provided in section 5 of this document.

#### Average Mode

The Avg_Mode enumeration is used to set the averaging mode applied to the raw data from the controller.

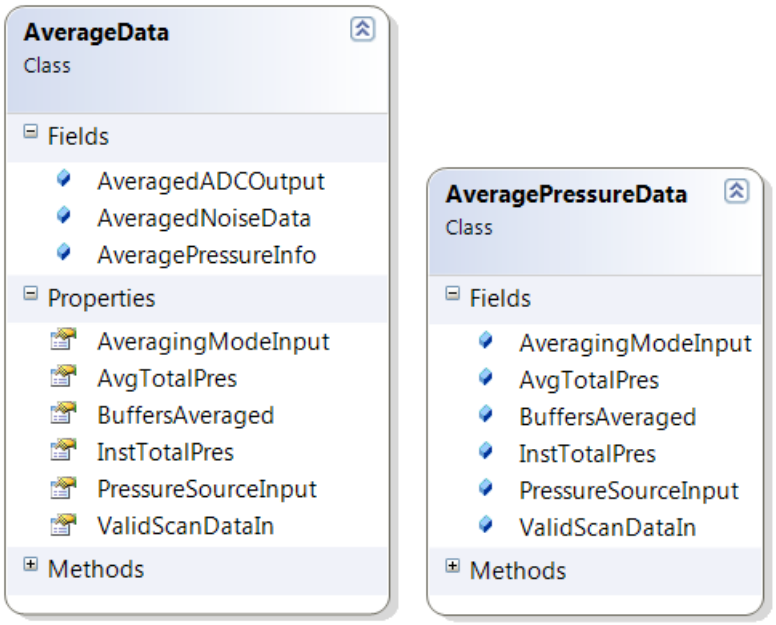



- **Off**: No averaging. The data is returned with no averaging.
- **Running_Avg** (AKA Finite Impulse Response (FIR) filter): The N most recent scans are summed and divided by N, where N is the number of scans to average. You may perform a running average on from 2 to 400 scans.
- **Cumulative_Moving_Avg** (AKA Infinite Impulse Response (IIR) filter): (N-1)*previous average + current scan)/N, where N is the number of buffers to average. You may perform a cumulative moving average from 2 to 50000 scans.
- **Accumulator**: Sum N scans and divide by N. A new value is reported every N scans. That is, unlike the Running Average, there are no intermediate results. The average data and analyzed data are only valid every N scans. The response time may be slow. You may perform an Accumulate average on from 2 to 50000 scans.

#### Pressure Source

Pressure sources can be connected to the 835 VQM Controller through the analog input connector on the front of the controller. Custom Pressure source configurations are retrieved from in the <install directory>\Data TotalPressureSources.dat file. Details for creating and using this file are documented in the 835 VQM Instruction Manual and the Customizing Instructions.txt file in the same directory. See Section 3.2.6 Total Pressure for the methods for retrieving information in this file.

**AverageData and AveragePressureData Classes**




double[] AveragedADCOutput- Averaged Counts as computed by the averager. This data is displayed in the Viewer Application on the Tune screen when Raw Counts are displayed - one point for each sample taken.

double[] AveragedNoiseData - Averaged array of samples of noise in counts.

AveragedNoiseData.Average() is used for the baseline noise when the scan range does not include 1.2 to 1.7 AMU.

Avg_Mode AveragingModeInput- Averaging mode applied to raw data, see Section 2.2.6, Average Mode.

double AvgTotalPres - Averaged Total Pressure in Torr. (The total pressure that is averaged is dependent on the Total Pressure Source. If the source is an analog input, the necessary conversion from voltage to pressure will occur before reporting.)

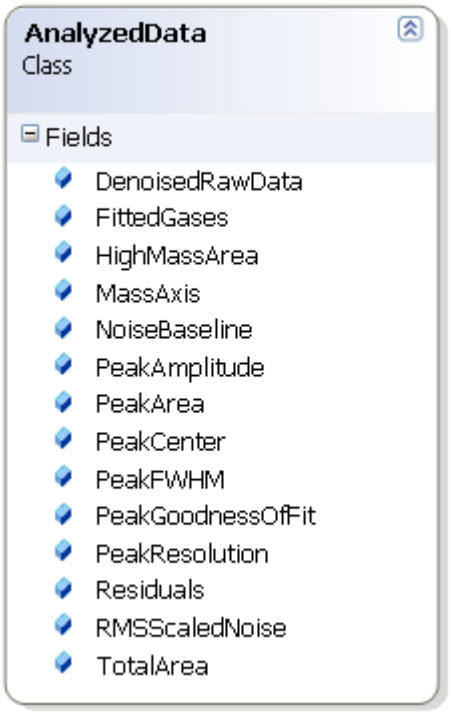
int BuffersAveraged - Number of samples collected so far.

double InstTotalPres - Instantaneous total pressure value in Torr. (The total pressure that is reported is dependent on the Total Pressure Source. If the source is an analog input, the necessary conversion from voltage (in Volts) to pressure will occur before reporting the pressure in Torr.)

string PressureSourceInput - Name/identifier for the Pressure source must match one of the strings returned from DataAnalysisGetPressureSourceStrings. See Section Pressure Source for more information.

bool ValidScanDataIn - Data is typically only invalid when Accumulator averaging is not yet ready or if the averager has just been reset.

**AnalyzedData Class**




The AnalyzedData Class stores the data after running through the data analysis routines. This is the main set of data that the 835 Viewer application uses to display results to the user.

double[] DenoisedRawData - Denoised intensity curve, analogous to Counts. Typically displayed as a curve as in the tuning curve on the Normalized view. One point for each sample taken.

double[] FittedGases - Gas Fractions (same scale as PeakArea). The factory provides 10 gases, although room exists for expansion to 30. See the System Manual for how to add coefficients for additional gases that may be in your vacuum system.

double[] MassAxis - Mass Axis in amu. One for each sample.

double NoiseBaseline – the value of the noise in counts that is subtracted from each sample.

double[] PeakAmplitude - The Amplitude of each mass peak. This is not equivalent to the amount of a particular mass. See PeakArea for the amount of a particular mass. One for each whole amu in the scan range.

double[] PeakArea - The Peak Area for each mass peak (equivalent to histogram). One for each whole amu in the scan range.
double[] PeakCenter - The Center value for each mass peak. One for each whole amu in the scan range.
double[] PeakFWHM - The Full Width Half Max of each mass peak. One for each whole amu in the scan range.

double[] PeakGoodnessOfFit - The Goodness of Fit for each mass peak. One for each whole amu in the scan range.

double[] PeakResolution - The Resolution for each mass peak. One for each whole amu in the scan range.

double[] Residuals - Residual Fractions as computed by gas fitting. One for each whole amu in the scan range. The same scale as PeakArea.

double RMSScaledNoise - Scaled RMS background noise.

# 3 CServiceWrapper Methods (.NET interface)

Programming examples for connecting to the VQM Controller are provided in section 5 of this document.

## 3.1 Connections

A user is allowed to use the API to connect to multiple VQM Controllers from a single application. Because more than one application can connect to a particular Controller, the Controller must be protected from two different applications sending it conflicting directions. The first application to connect to a Controller has "Master" privileges. i.e., it has permission to store setpoint values, change operating mode, etc. Any other application that connects to the Controller has "Monitor" privileges. i.e., they can read header values, scan data, min/max values, setpoints, etc., but cannot change the values. They also can not turn on or off the gauge or scanning.

Only when all connections to a Controller are closed, will the next connection be assigned as Master. Note that it is not sufficient to disconnect just a Master. If a monitor is connected, the next connection will be a monitor even if no master exists. Similarly, a monitor is not promoted to a master when a master disconnects.

The available Controllers list returned by GetValidConnections is retrieved at the time of the call and can become invalid due to events. For example, a user could disconnect the USB cable after the list is generated and returned. If the user tries to connect to a Controller that is not available, an error will be returned. It may be necessary for the user to unplug the USB cable, cycle power on the Controller, and/or restart the services to recover.

### 3.1.1 GetValidConnections

Returns a list of VQM Controllers connected to COM ports.

```
VQM_800_ERROR GetValidConnections(out CDeviceConnectionInfo[] connectionInfoList,
      CDeviceConnectionInfo.EServiceConnectionType serviceConnectionType =
      CDeviceConnectionInfo.EServiceConnectionType.LOCALHOST,
      string serviceConnection = "")
```

Note that the last two parameters have default values for the normal case of getting the valid connections for the local machine. The other parameters are used if you are getting the valid connections for a remote computer – see Special Considerations for Remote Connections 1.2 .

On return, the connectionInfoList array will contain a list of CDeviceConnectionInfo that identify the Controllers.

### 3.1.2 ConnectToDevice

Connects to a specific VQM Controller using the input connection information. Normally this would be returned by a call to GetValidConnections.

```
VQM_800_ERROR ConnectToDevice(CDeviceConnectionInfo connectInfo, out bool
isMaster);
```

Attach this process to the particular Controller identified by the CDevideConnectionInfo parameter.

If this command succeeds, then it is the user's responsibility to save the connectInfo variable information to be used with all other commands. The CDeviceConnectionInfo values provide an application with a method of maintaining separate connections to multiple controllers and differentiating among them. The isMaster parameter will let the caller know if the connection was made as a master or a monitor.

The first application to open a connection to a Controller will become the Master. A Master has permission to change settings on the Controller. Any subsequent connection to the same Controller will have the type Monitor. A Monitor cannot change settings or the state of the Controller, but may read any returned data.

### 3.1.3 DisconnectFromDevice

Disconnects from and closes the connection and service to a referenced controller.

```
VQM_800_ERROR DisconnectFromDevice(CDeviceConnectionInfo connectInfo);
```

If there are more connections than just this one, the remaining connections are maintained. If this is the last connection to be released, the Master connection is made available for the next connection. DisconnectFromDevice is protected against threading issues.

### 3.1.4 ConnectToRawData

Opens a connect much like ConnectToDevice, but to a raw log file.

```
VQM_800_ERROR ConnectToRawData(string fullPath, CDeviceConnectionInfo connectInfo,
RawConnectionOptions options = null);
```

```
Where connectinfo.DeviceType
=CDeviceConnectionInfo.EconnectionType.RAW_DATA_LOG_835.
```

The options parameter allows configuration of how the calls to GetScanDataFromLog will respond. Defaults exist for all of the possible configurable parameters.

### 3.1.5 CloseConnection

Disconnects all controllers and cleans up all Service connections and event handling.

```
public void CloseConnection();
```

This command needs to be called when exiting the application. Failure to do so may cause an exception being thrown.

## 3.2 Configuration

### 3.2.1 Controller Non-Volatile RAM

The controller has Non-Volatile memory. These commands interact with it. These commands will not return from the service until they are complete. Calling applications will need to get data returned, or handle a timeout for these commands. A monitor will be blocked from fetching scan/header data during these commands.

Programming examples to restore factory or user defaults are provided in section 5 of this document.

**RestoreFactoryDefaults**

Restores the factory defaults from the non-volatile memory.

```
VQM_800_ERROR RestoreFactoryDefaults(CDeviceConnectionInfo connectInfo);
```

Read the Factory setpoints from non-volatile memory and store them as the current setpoints. Note that the Controller will remove gauge power before performing this activity. The user will have to turn the gauge on again. Gauge power is not restored automatically.

**RestoreUserSettings**

Restores the user settings from the non-volatile memory on the controller.

```
VQM_800_ERROR RestoreUserSettings(CDeviceConnectionInfo connectInfo);
```

Read the User setpoints from non-volatile memory and store them as the current setpoints. Note that the Controller will remove gauge power before performing this activity. The user will have to turn the gauge on again. Gauge power is not restored automatically.

**SaveUserSettingsToEEPROM (Store User Settings)**

Stores the current setpoints to non-volatile memory as the User Settings.

```
VQM_800_ERROR SaveUserSetttingsToEEPROM(CDeviceConnectionInfo connectInfo);
```

Store the active user setpoints to non-volatile memory as the default user setpoints. Restoring User Settings or powering on will restore these setpoints. Note that the Controller will remove gauge power before performing this activity. The user will have to turn the gauge on again. Gauge power is not restored automatically.

### 3.2.2 Controller Operating Parameters

**GetExtRangeCapabilities**

Discover the range capabilities of the attached Controller.

```
VQM_800_ERROR GetExtendedRangeCapabilities(out bool isExtendedRange,
CDeviceConnectionInfo);
```

The output parameter "isExtendedRange" is true if the Controller supports extended range scanning (approximately 1-300 amu) and false if it doesn't (approximately 1-145 amu). The ranges are approximate because the calibration changes the range slightly.

**GetScanRange**

Get the current scan range of the controller. This is the actual range it is configured to scan, not its full range.

```
VQM_800_ERROR GetScanRange(out double lowerRange, out double upperRange,
CDeviceConnectionInfo connectInfo);
```

Both monitors and masters can read the current scan range. The range your Controller supports depends on the product you ordered. See GetExtRangeCapabilities above.

**SetScanRange**

Sets the scan range for the Controller. Monitors can not successfully use this call.

```
VQM_800_ERROR SetScanRange(ref double lowerRange, ref double upperRange,
CDeviceConnectionInfo connectInfo);
```

The lowerRange and upperRange parameters are transmitted to the Controller for use as the scan limits. A Controller Error may result if the range is not supported by the controller.

**GetMinMaxMassCalibrationFactor**

Returns the minimum and maximum allowed values for the Mass Calibration Factor.

```
VQM_800_ERROR GetMinMaxMassCalibrationFactor(out CLogicalInstrumentMinMax minMax,
CDeviceConnectionInfo connectInfo);
```

Populates parameter "minMax" with the minimum and maximum values.

**GetMassCalibrationFactor**

Returns the most recent value for the Calibration Factor. **The mass calibration factor is a float, not a double.**

```
VQM_800_ERROR GetMassCalibrationFactor(out float value, CDeviceConnectionInfo
connectInfo)
```

The parameter "value" is populated with the current calibration factor from the Controller.

**SetMassCalibrationFactor**

Sets the value of the Calibration Factor. **The input value is a float, not a double.** Monitors can not successfully use this call.

```
VQM_800_ERROR SetMassCalibrationFactor(float value, CDeviceConnectionInfo
connectInfo);
```

The parameter "value" is transmitted to the Controller for use as the calibration factor.

**GetElectrometerGainMinMax**

Returns the minimum and maximum allowed value for the Electrometer Gain.

```
VQM_800_ERROR GetElectrometerGainMinMax(out double minValue, out double maxValue,
CDeviceConnectionInfo connectInfo)
```

This command populates the minimum and maximum parameters from the Controller. These values do not change after they are initialized, so no additional communications are required to return these values. Similarly, a user needs only fetch them once per Controller.

**GetElectrometerGainSetpoint**

Returns the current setpoint for the Electrometer Gain.

```
VQM_800_ERROR GetElectrometerGainSetpoint(out double value, CDeviceConnectionInfo connectInfo);
```

The parameter “value” is populated with the current electrometer gain setpoint from the controller.

**GetElectrometerGain**

Returns the electrometer gain as measured when it was originally set. **The value cannot be measured during normal operation. If a master makes this call, the gauge will be turned off to read the value. The gauge state will be restored after the command is complete. Monitors should use GetElectrometerGainSetpoint.**

```
VQM_800_ERROR GetElectrometerGain(out double value, CDeviceConnectionInfo connectInfo);
```

Populate parameter "value" with the value of the electrometer gain (measured).

**SetElectrometerGainSetpoint**

Sets the setpoint for the Electrometer Gain. Monitors can not successfully use this call. **Note that the call will result in the gauge's being shut off, the value changed, and then the gauge state restored.**

```
VQM_800_ERROR SetElectrometerGainSetpoint(double value, CDeviceConnectionInfo connectInfo);
```

The parameter "value" is transmitted to the Controller for use as the electrometer gain target. Wait several seconds before calling GetElectrometerGain to retrieve an accurate result.

**GetFilamentEmissionCurrentMinMax**

Returns the minimum and maximum allowed value for the Filament Emission Current.

```
VQM_800_ERROR GetFilamentEmissionCurrentMinMax(out double minValue, out double maxValue, CDeviceConnectionInfo connectInfo)
```

The parameters are populated with the minimum and maximum setpoints for Filament Emission Current. These values are defined by the Controller.

**GetFilamentEmissionCurrentSetpoint**

Returns the current setpoint for the Filament Emission Current.

```
VQM_800_ERROR GetFilamentEmissionCurrentSetpoint(out double value, CDeviceConnectionInfo connectInfo);
```

The parameter "value" is populated with the Filament Emission Current Setpoint.

**GetFilamentEmissionCurrent**

Returns the current Filament Emission Current.

```
VQM_800_ERROR GetFilamentEmissionCurrent(out double value, CDeviceConnectionInfo connectInfo);
```

The parameter "value" is populated with the Filament Emission Current.

**SetFilamentEmissionCurrentSetpoint**

Sets the setpoint for the Filament Emission Current. Monitors can not successfully use this call.

```
VQM_800_ERROR SetFilamentEmissionCurrentSetpoint(ref double value,
CDeviceConnectionInfo connectInfo);
```

The parameter "value" will contain the current setpoint after the new value is stored.

**GetLogicalInstrumentMinMaxVoltage**

Returns the minimum and maximum allowed values for the input logical instrument.

```
VQM_800_ERROR GetLogicalInstrumentMinMaxVoltage(Utilities.ELogicalInstruments
logicalInstrumentEnum, out CLogicalInstrumentMinMax minMax, CDeviceConnectionInfo
connectInfo)
```

The parameter minMax is populated with the minimum and maximum setpoints for the particular logical instrument's voltage (in Volts) setpoint from the Controller.

Attempts to set the Logical Instrument's voltage outside of this range will result in a Controller error.

**GetLogicalInstrumentVoltageSetpoint**

Returns the current setpoint for the input logical instrument in Volts.

```
VQM_800_ERROR GetLogicalInstrumentVoltageSetpoint(Utilities.ELogicalInstruments
logicalInstrumentEnum, out double value, CDeviceConnectionInfo connectInfo);
```

The parameter "value" is populated with the requested logical instrument's voltage setpoint.

**GetLogicalInstrumentCurrentVoltage**

Returns the current voltage for the input logical instrument in Volts.

```
VQM_800_ERROR GetLogicalInstrumentCurrentVoltage(Utilities.ELogicalInstruments
logicalInstrumentEnum, out double value, CDeviceConnectionInfo connectInfo);
```

The parameter "value" is populated with the current value of the requested logical instrument's voltage.
*Note, this value is not the same as the setpoint for the voltage. It is the measured voltage. Measured voltage depends on gauge state. If you have recently changed gauge state, you might want to wait several seconds to read the current voltage. Optionally, use GetHeaderData or GetScanData to read current voltages if you are reading values for several parameters.*

**SetLogicalInstrumentVoltageSetpoint**

Sets the voltage setpoint for the input logical instrument in Volts. Monitors can not successfully use this call.

```
VQM_800_ERROR SetLogicalInstrumentVoltageSetpoint(Utilities.ELogicalInstruments
logicalInstrumentEnum, ref double value, CDeviceConnectionInfo connectInfo);
```

### 3.2.3 Auto Tune

**StartAutotune**

To initiate Auto Tune, you must have a master connection and Auto Tune can not already be running.

```
VQM_800_ERROR StartAutotune(string logFileName, CDeviceConnectionInfo connectInfo,
EAutotuneType mode = EAutotuneType.NORMAL, EAutotuneEMBiasTargetPeakHeight
targetPeakHeight = EAutotuneEMBiasTargetPeakHeight.Normal_20na)
```

Initiates either a normal or Detailed Auto Tune procedure. All other activity with this controller will be locked out until Auto Tune completes or is cancelled. The auto tune log will be written to logFileName. The path (but not the file) must exist.

**StartExpressAutotune**

Initiates an Express Auto Tune procedure. Express Autotune only sets the filament emission current appropriately for the total pressure range (if the total pressusre is available) and adjusts the EM Bias voltage gain so that the maximum peak in the normal scan range of 1-135 AMU is at the targeted level. TargetedPeakHeight are approximately 8 nAmps for long life, approximately 20 nAmps for

standard scanning, and approximately 50 nAmps for high sensitivity. The default targetPeakHeight is for normal approximately 20 nAmps.

```
VQM_800_ERROR StartExpressAutotune(string LogFileName, CDeviceConnectionInfo
connectInfo, EAutotuneEMBiasTargetPeakHeight targetPeakHeight =
EAutotuneEMBiasTargetPeakHeight.Normal_20na)
```

**StartJustEMAutotune**

Initiates an EM Only Auto Tune procedure to tune the EM Bias gain so that the highest peak in the current scan range is approximately the targeted peak height. This procedure is useful if the scan range is narrower than the standard range or if trap parameters such as Repeller, Exit plate, EM Shield voltage or filament emission current have been successfully adjusted manually for the vacuum environment.

```
VQM_800_ERROR StartJustEMAutotune(string LogFileName, CDeviceConnectionInfo
connectInfo, EAutotuneEMBiasTargetPeakHeight targetPeakHeight =
EAutotuneEMBiasTargetPeakHeight.Normal_20na)
```

**GetAutotuneStatus**

GetAutoTuneStatus provides the complete status of the current or just-completed Auto Tune procedure. If Auto Tune completed successfully, the values for each of the tuned parameters is available in the in the returned values as well as on the controller. The values have not been stored on the controller so they can be lost in the event of a reset or power-cycle.

```
VQM_800_ERROR GetAutoTuneStatus(out EAutotuneStatus status, out int percentDone,
out double emBias, out double rep, out double explate, out double shield, out
double rfamp, out double filCurrent)
```

Returns the current status of the Auto Tune procedure including percent done, last error, Auto Tuned values for EM Bias voltage, Repeller, Exit Plate, EM Shield, and Filament Emission Current as output parameters. Note that the voltages are not valid unless Auto Tune has completed and succeeded. If Auto Tune has failed, the voltages are all zeroes. Voltage units are in Volts, current in nanoAmps.

**CancelAutotune**

Use this call to halt an active Auto Tune procedure. You must be master to call this function. However, you do not need to be actively Auto Tuning to Cancel. The call will not only terminate the procedure, but will reset the outputs.

```
VQM_800_ERROR CancelAutotune(CDeviceConnectionInfo connectInfo);
```

This function can be used to terminate an active Auto Tune procedure or reset a completed one. It is not an error to call AutoTuneCancel without an active Auto Tune procedure.

### 3.2.4 Averaging

**DataAnalysisClearAvgHistory**

Clears the average history of all previous samples.

```
VQM_800_ERROR DataAnalysisClearAvgHistory(CDeviceConnectionInfo connectInfo);
```

This method clears the averaging history. It should be called any time the mode changes. If the current number of buffers collected has exceeded the new number of buffers to average, it should be called also. The current number of averages will be reset when this method is called.

**DataAnalysisSetAvgMode**

Sets the average mode for the averager in the data analysis.

```
VQM_800_ERROR DataAnalysisSetAvgMode(Avg_Mode avgMode, CDeviceConnectionInfo
connectInfo);
```

Set the mode used for Averaging. The current number of averages will be reset when the average mode is changed.

**DataAnalysisSetNumAvgs**

Sets the number of data sets to average. This interacts with the averaging mode.

```
VQM_800_ERROR DataAnalysisSetNumAvgs(int numAverages, CDeviceConnectionInfo
connectInfo);
```

Set the number of buffers used for averaging. Each mode has its own min and max limits. Averaging Mode None has min and max of 1. The running average has min of 2 and max of 100. The others have min of 2 and max of 50000. The current number of averages will be reset when the mode is changed or the current number of buffers to be averaged is smaller than the new number of buffers to be averaged.

### 3.2.5 Data Analysis

**DataAnalysisSetHighMassIndex**

This method sets the mass for which the HighMassArea will be calculated. It is recommended not to use this method. The highMass input to this function is an index in the mass axis for the mimimum mass that is considered the high mass. As a result it change swith the mass axis and mass calibration factor. Instead use the high mass Amu method explained in the system manual for the new easier, more intuitive method.

```
VQM_800_ERROR DataAnalysisSetHighMassIndex(int highMass, CDeviceConnectionInfo
connectInfo);
```

HighMassArea is equal to the area under the curve (total charge) for all the scanned data from the HighMassIndex (index into the x-axis table) to the end of the scan range. Some laboratories define all mass above 40 amu to be organic compounds. The value of HighMassArea represents this value. Its value will include doubly-ionized peaks that are not found via the normal peak finding because they are at half-amu locations between whole amu peaks. The PeakArea calculations for amus above the HighMassIndex are still calculated and populated normally.

DataAnaysisSetResolutionLimits

Sets the high and low resolution limits for peak fitting. If peaks are seen but not fitted the resolution limits may need to be customized. If the RF has been set higher than 0.5 the resolution the default resolution minium of 70 may need to be lowered. The default resolution maimumis 300 to eliminate noise spikes, bu thtis can also be adjusted up to 1000. In general, however, the resolution limits should be left alone.

```
VQM_800_ERROR DataAnalysisSetResolutionLimits(double highResolutionLimit, double
lowResolutionLimit)
```

### 3.2.6 Total Pressure

Total Pressure is derived from many sources. Each application may have its own display units. Therefore, the Service will return all pressures in Torr. Each application can apply desired units conversion. GetHeaderData will continue to include the Total Pressure as reported by the 390802 (if such exists), and its units will be in the 390802 units.

Analog IN is in the header and will contain the analog voltage, in Volts, corresponding to the pressure from an analog pressure gauge connected to the controller. The Analog IN field will always report the voltage input to the Analog IN BNC of the VQM Controller.

**DataAnalysisGetPressureSourceStrings**

Gets a string array of pressure sources. The TotalPressureSources.dat file can be used to add pressure sources that are not already included in the software as documented in the 835 VQM Instruction Manual (p/n 835000).

```
VQM_800_ERROR DataAnalysisGetPressureSourceStrings(out string[] pSources)
```

**DataAnalysisSetPressureSource**

Sets the pressure source. The setting is used when calculating instantaneous total pressure and average total pressure.

```
VQM_800_ERROR DataAnalysisSetPressureSource(string source, CDeviceConnectionInfo
connectInfo)
```

Set the Total Pressure Source for use by GetExternalTotalPressure. Source is a string identifying the pressure source in use and needs to match a string the inte list of pressure sources returned from DataAnalysisGetPressSoruceStrings. The source information is used to convert either the 390802's reported pressure (and its programmed units) or the Analog Input Voltage to Torr.

**GetExternalTotalPressure**

Gets the current value from the external total pressure sensor.

```
VQM_800_ERROR GetExternalTotalPressure(out double value, CDeviceConnectionInfo
connectInfo)
```

The parameter "value" is populated with the Total Pressure.

If the Total Pressure Source is the 390802 TPMK, the value will be the Total Pressure reported by the 390802 in Torr. If the Total Pressure Source is any of the Analog Input gauges, the value will be the Analog Input value converted appropriately to Torr. If the Total Pressure Source is None, the value will be 0. If the value is below 1E-15 (independent of the source), it will be reported as 0.

Note that the 390802 Total Pressure is reported in Torr, not the units of the 390802. A user who wants to read total pressure in other units should wrap GetExternalTotalPressure in a user-defined function to perform the conversion. The reason is that the 390802 is the only Total Pressure source that has its own units. The translation equations for all of the others assume Torr. Rather than have the user deal with changing from Torr to 390802 units and back each time the user changes the total pressure source, the API always returns the Total Pressure and partial pressures in Torr. It is the user's responsibility to convert to display units.

### 3.2.7 Errors

**GetErrorCountAndQueue**

Gets the errors from the Controller error queue. The errors will be cleared from the queue after they have been returned. The length of the errorQueue array should be equal to the errorCount, but a careful programmer may choose to use errorQueue.Length rather than the count.

```
VQM_800_ERROR GetErrorCountAndQueue(out int errorCount, out string[] errorQueue,
CDeviceConnectionInfo connectInfo);
```

After each logical sequence associated with an API call that accesses the Controller, and after any Service-periodic sequence (for example Fetch), both the “errorCount” and “errorQueue” will be read from the Controller and information related to them are stored for each current connection to the associated device. Updates will be in the form of an Append. That is, entries are added to the connection's queue for each entry in the Controller's ErrorQueue. Limitations may apply. For example, the Controller's error queue holds only 20 entries. In the cases where the queue would overflow before being read by the user, the last entry indicates a queue overflow.

Of particular interest to users who intend to support more than one connection to a single controller (for example, one master and one or more monitors), the only API call that will return a CONTROLLER_ERRORS response is the call for which the errors were first detected. This call may be associated with either a master or a monitor. Therefore, API users will want to check the error count and queue to detect the errors that were generated by a different connection. Please note, a master could miss the call just as easily as a monitor. Any application should expect to check the error count and queue for thoroughness.

### 3.2.8 Header Data, Scan Data, Analysis Results

These calls are frequently used periodically. For example, if the normal scan time is 80 milliseconds, a user application may be designed with a timer to call GetScanData every 80 milliseconds.

**GetHeaderData**

Returns the most recent Header Data.

```
VQM_800_ERROR GetHeaderData(out HeaderData835 stateData, CDeviceConnectionInfo
connectInfo);
```

The parameter "stateData" is populated with values returned from the Controller via the Fetch command.

**GetHeaderData (with some average data)**

Returns the most recent Header Data and some valid averaging parameters. The method is only applicable when the GaugeState is anything other than SCAN.

Programming examples to get header data are provided in section 5 of this document.

```
VQM_800_ERROR GetHeaderData(out int AvgBuffersAveraged, out double AvgTotalPres,
out double AvgInstTotalPres, out string AvgPressureSourceInput, out Avg_Mode
AveragingModeInput, out bool ValidScanDataIn, out HeaderData835 currentStateData,
CDeviceConnectionInfo connectInfo)
```

The parameter "currentStateData" is populated with values returned from the Controller via the Fetch command. Other output values are the settings for averaging and the total pressure (instantaneous and averaged.) Note the header values are valid as long as there is a connection to a Controller. It is not required that the Controller be scanning. ValidScanDataIn indicates whether the average data output parameters are valid. The most likely reason why they would not be valid is that the Averaging Mode is Accumulate and the number of buffers has not yet been accumulated. **Do not use the averaging outputs unless the ValidScanDataIn parameter is true.**

**GetProcessedScanData (structured)**

Gets the current scan data. This consists of Analyzed Results, Averaged Pressure Data, and a Scan Number. The Scan Number may be used to detect duplicate or missed scans.

```
VQM_800_ERROR GetProcessedScanData(ref int scanNumber, out AnalyzedData results,
out AveragePressureData averagePressureData, CDeviceConnectionInfo connectInfo)
```

The set of output parameters represents a snapshot of the results data after a particular scan number. To use this method, the parameter scanNumber should be set to the last value returned (the last time the method was called.) Initialize it to 0 for the first call. The result of the call should be either an error or a new scan number.

**GetScanData (unstructured)**

Gets the current scan data. This consists of Header Data, Analyzed Results, Averaged Data, and a Scan Number. The Scan Number may be used to detect duplicate or missed scans.

```
VQM_800_ERROR GetScanData(ref int scanNumber, out double[] FittedGases, out
double[] Residuals, out double[] MassAxis, out double[] DenoisedRawData, out
double[] PeakArea, out double[] PeakGoodnessOfFit, out double[] PeakFWHM, out
double[] PeakResolution, out double[] PeakAmplitude, out double[] PeakCenter, out
double RMSScaledNoise, out bool AvgValidScanDataIn, out int AvgBuffersAveraged,
out double AvgTotalPres, out double AvgInstTotalPres, out string
AvgPressureSourceInput, out double[] AveragedADCOutput, out double TotalArea, out
double HighMassArea, out double NoiseBaseline, out Avg_Mode AveragingModeInput,
out HeaderData835 currentStateData, CDeviceConnectionInfo connectInfo, out bool
isValidData, out double[] AveragedNoiseData, out EGaugeState controllerState)
```

The set of output parameters represents a snapshot of the results data after a particular scan number. To use this function, the parameter scanNumber should be set to the last value returned (the last time the method was called.) Initialize it to 0 for the first call. The result of the call should be either an error or a new scan number.

More explicitly, the sum of these output parameters represents a cohesive entity called "results". Individual "get" functions do not maintain the cohesiveness of the results. For example, you should not assume that a Total Pressure returned by GetExternalTotalPressure can be used to calculate partial pressures from this results set. Instead, use the total pressure returned by this call.

Not everyone will need all output parameters from this call. The table below explains what analog the parameter has in the Viewer Application.

**Table 1: Correlation between Parameters received from GetScanData and Viewer Application**

| Parameter | Interpretation | Display In UI |
|---|---|---|
| FittedGases | An array of the fractions of the fitted gases. Fractions = %/100. | Shows in the results table in "Gas" mode, converted to %, and converted to partial pressure. These are labeled with associated gas names. |
| Residuals | An array of the fractions of residual gases. Fractions = %/100. | Shows in the results table in "Gas" mode, converted to %, and converted to partial pressure. These are labeled with just the amu. |
| MassAxis | This is the amu or "X" values associated with the raw data. | Used to display the raw data with the correct x-axis |
| DenoisedRawData | Data has been averaged and denoised. | Displayed in the tuning graph when in Normalized display mode. |
| PeakArea | Area under the fit curve. Units are Fractions = %/100. | Displayed in Selected Peak Parameters for selected peak as "Intensity" and in the results table if displayed in "Mass" mode instead of "Gas" mode. Used for Histogram values. |
| PeakGoodnessOfFit | Goodness of fit for the fit curve | Displayed in Selected Peak Parameters for selected peak as "GoF" |
| PeakFWHM | FWHM for fit curve | Displayed in Selected Peak Parameters for selected peak as "FWHM" |
| PeakResolution | Resolution for fit curve | Displayed in Selected Peak Parameters for selected peak as "Resolution" |
| PeakAmplitude | Amplitude for fit curve | Not directly displayed in interface, but used to calculate the Signal to Noise Ratio, displayed as "SNR". Do not confuse with PeakArea. |
| PeakCenter | Center of fit curve | Displayed in Selected Peak Parameters for selected peak as "Location" |
| RMSScaledNoise | RMS scaled noise calculated by analysis | Not directly displayed in interface, but used to calculate the Signal to Noise Ratio, displayed as "SNR" |
| AvgValidScanDataIn | Boolean to indicate wheter AverageData is valid. | If AvgValidScanDataIn is false, do not use the Averaging Data. |
| AvgBuffersAveraged | The number of samples included in the current average. | Displayed as "Avgs Collected" in the Post-Processing section. |
| AvgTotalPres | This is the averaged value for the total pressure reading. Always returned as Torr. Convert as desired. | Displayed as "Average Total Pressure" but may be scaled to display units |

| Parameter | Interpretation | Display In UI |
|---|---|---|
| AvgInstTotalPres | This is the instantaneous value for the total pressure reading of the most recent sample. Always returned as Torr. Convert as desired. | Displayed as "Current Total Pressure" but may be scaled to display units |
| AvgPressureSourceInput | The Pressure Source | This is equivalent to the "Pressure Source" selector on the settings page. |
| AveragedADCOutput | Data has been averaged but not denoised. | Displayed in the tuning graph when in Raw display modes. It is shown in counts, or rescaled to show in nA. |
| TotalArea | The total charge represented by the sum of the areas under the peaks. | |
| HighMassArea | The total charge above the high mass AMU set. Note this will include charges at fractional AMUs. Typically this is used to identify the ratio of organics in the vacuum system. | |
| NoiseBaseline | The value of the noise in counts that is subtracted from each sample. | |
| AveragingModeInput | The averaging mode | This is equivalent to the "Averaging Mode" selector in the Post-Processing section. |
| currentStateData | This is the Header data returned in the scan which corresponds to the Header display on the Tune Screen of the Viewer Application. It includes the actual values of the trap settings, the pressure and pressure units from the 390802 connected to the controller via RS-485 (if present), the analog in voltage , and the serial number and firmware revision of the controller. | See the Tune Screen Header section when select diagnostics display. |
| connectInfo | The reference to the connection (of type CDeviceConnectionInfo) used in almost all API calls. | |
| isValidData | Boolean to indicate whether or not fresh valid scan data was received and averaged. When the averaging mode is set to accumulator, this will be false until the number of samples for averaging is collected. If isValidData is false, the returned data should not be used. | |

| Parameter | Interpretation | Display In UI |
|---|---|---|
| AveragedNoiseData | Averaged array of samples of noise in counts. AveragedNoiseData.Average() is used for the baseline noise when the scan range does not include 1.2 to 1.7 AMU. | |
| controllerState | Gauge state as defined by EGaugeState enumeration | Gauge state icon is displayed on the UI next to the “Connect” button after a controller is connected. |

**GetScanData (structured)**

Similar to GetScanData (unstructured), but with the results packed into more complex data structures.

```
VQM_800_ERROR GetScanData(ref int scanNumber, out AnalyzedData results, out
HeaderData835 currentStateData, out AverageData averageData, CDeviceConnectionInfo
connectInfo, out bool isValidData, out EGaugeState controllerState)
```

See section 2.2.6 for Data Analysis classes and Table 1: Correlation between Parameters received from GetScanData and Viewer Application contains a description of the values in the structures as used in the Viewer Application.

**GetScanData (unstructured)**

Similar to GetScanData (structured)

Gets the current scan data. This consists of Analyzed Results, Averaged Pressure Data, and a Scan Number. The Scan Number may be used to detect duplicate or missed scans.

The set of output parameters represents a snapshot of the results data after a particular scan number. To use this method, the parameter scanNumber should be set to the last value returned (the last time the method was called.) Initialize it to 0 for the first call. The result of the call should be either an error or a new scan number.

GetScanData (unstructured)

```
VQM_800_ERROR GetScanData(ref int scanNumber, out AnalyzedData results, out
HeaderData835 currentStateData, out AverageData averageData, CDeviceConnectionInfo
connectInfo, out bool isValidData, out EGaugeState controllerState)
```

See AnalyzedData details and AverageData. Table 1: Correlation between Parameters received from GetScanData and Viewer Application contains a description of the values in the structures as used in the Viewer Application.

**GetScanDataFromLog (unstructured)**

Much like GetScanData (unstructured), this function returns all the same fields, but from the connected log file.

```
VQM_800_ERROR GetScanDataFromLog(ref int scanNumber, out double[] FittedGases, out
double[] Residuals, out double[] MassAxis, out double[] DenoisedRawData, out
double[] PeakArea, out double[] PeakGoodnessOfFit, out double[] PeakFWHM, out
double[] PeakResolution, out double[] PeakAmplitude, out double[] PeakCenter, out
double RMSScaledNoise, out bool AvgValidScanDataIn, out int AvgBuffersAveraged,
out double AvgTotalPres, out double AvgInstTotalPres, out string
AvgPressureSourceInput, out double[] AveragedADCOutput, out double TotalArea, out
double HighMassArea, out double NoiseBaseline, out Avg_Mode AveragingModeInput,
out HeaderData835 currentStateData, CDeviceConnectionInfo connectInfo, out bool
endOfLogData, out double[] AveragedNoiseData)
```

When the end of the logged data is reached, the Boolean endOfLogData is true.

**GetScanDataFromLog (structured)**

Much like 3.2.8.5 GetScanData (structured) except the data is from a previously saved raw log file instead of real-time.

```
VQM_800_ERROR GetScanDataFromLog<T>(ref int scanNumber, out AnalyzedData results,
out T currentStateData, out AverageData averageData, CDeviceConnectionInfo
connectInfo, out bool isValidData, out bool endOfLogData) where T : FetchHeader
```

When the end of the logged data is recahced, endOfLogData is true.

### 3.2.9 Analog and Digital I/O

On the standalone controller, the Analog and Digital outputs have a special use: the Analog Output is the raw scan curve and the Digital Output is a framing pulse for the Analog Output. The Digital Output has an alternative use: it can be software controlled. However, when the Digital Output is controlled by the software, its mode is changed from framing pulse to software-controlled digital output. If you Save User settings after setting the digital output high or low via the software, your mode and value settings will be saved also. Restore User settings via command or power cycle will restore the Digital Output's mode and state. You can recover the use of the Digital Output to the framing pulse by restoring factory defaults or by sending a SCPI command to change the Digital Outputs mode.

**GetAnalogInputVoltage**

Returns the current value of the external signal voltage. If you are using the Analog Input as a total pressure input, configure Total Pressure Source and use GetExternalTotalPressure (0) instead. Use this function if you have connected some other Analog Input device.

```
VQM_800_ERROR GetAnalogInputVoltage(out double value, CDeviceConnectionInfo
connectInfo);
```

Populate parameter "value" with the voltage from the Analog Input, in Volts.

**GetDigitalOut**

Get the current value of the Digital Output.

```
VQM_800_ERROR GetDigitalOut(out bool digitalOutState, CDeviceConnectionInfo
connectInfo);
```

Return the current value of the Digital Out (labeled Trigger Out on the Controller).

**SetDigitalOut**

Sets the Digital Output (labeled Trigger Out on the Controller) to the requested state.

```
VQM_800_ERROR SetDigitalOut(bool digitalOutState, CDeviceConnectionInfo
connectInfo);
```

Changes the operating mode of the Digital Output from Framing Pulse to Digital. If digitalOutState is true, the output is driven High. If digitalOutState is false, the output is driven Low. Once the output mode is set to Digital Out, it can not be reset to Framing Pulse with this command. See the table below.

| Current Output Mode | SetDigitalOut | Next Output Mode |
|---|---|---|
| Framing Pulse | High | Digital Output |
| Framing Pulse | Low | Framing Pulse |
| Digital Output | High | Digital Output |
| Digital Output | Low | Digital Output |

There is no API command to change the output mode back to Framing Pulse once Digital Output is chosen. The output mode is saved to the non-volatile user settings when StoreUserSettings is called. If the Controller is

power-cycled, digitalOutState will set it to the last saved state. There is a SCPI command to restore the setting. See an example in 3.3 SendInstruction.

### 3.2.10 Gas-Fitting Definitions

**GetComboFile (unconnected)**

Gets the coefficients used for gas-fitting and residual calculations with no connection. That is, you do not need connection info to get this information.

```
VQM_800_ERROR GetComboFileData(out ComboFileData data);
```

ComboFileData is an array of objects, each of which includes a gas name and 4 arrays, two each for the fitting profile and the residual calculation profile. The two arrays are one for amu values and one for the relative intensity of the expected amu's signal. These are analogous to the histograms provided for gases by NIST, but the values are different because the 835 gauge's emission current is different from that used in the NIST calculations. A user will get the most value from knowing the names associated with each gas. The order of the objects in the ComboFileData returned is the same as the order in FittedGases from GetScanData.

**GetComboFileData (connected)**

Gets the coefficients used for gas-fitting and residual calculations for a connection. No connection information is needed to get this information, but this version of the call is provided to support the common format.

```
VQM_800_ERROR GetComboFileData(out ComboFileData data, CDeviceConnectionInfo
connectInfo);
```

See the description above.

### 3.2.11 Logging

**GetLogFileFolderPath**

```
VQM_800_ERROR GetLogFileFolderPath(out string folder, CDeviceConnectionInfo
connectInfo);
```

This method provides the current raw log file path. The path might be used to pre-populate a dialog that prompts the user to enter a raw log file path.

**SetLogFileFolderPath**

```
VQM_800_ERROR SetLogFileFolderPath(string folder, CDeviceConnectionInfo
connectInfo);
```

This method provides a way for the application to change the destination where raw log files are written. (The installer creates a default path for the raw log file in the public documents area.) The calling application must create the folder sent in the folder string if it does not already exist and the path sent must be an absolute path. Consider the StartLogging method, which can also specify the folder path.

**GetLogFileScanCount**

Returns the number of scans recorded in each Log File.

```
VQM_800_ERROR GetLogFileScanCount(out int value, CDeviceConnectionInfo
connectInfo);
```

The value is the number of scans that will be written to each of the serialized Log Files. Log Files are serialized to make it easier to open them with various viewers and editors. Log files can get very large. Each line is long and there are many lines.

**SetLogFileScanCount**

Sets the number of scans that will be recorded in each Log File before starting the next serialized file.

```
VQM_800_ERROR SetLogFileScanCount(int scanCount, CDeviceConnectionInfo
connectInfo);
```

**StartLogging**

Starts logging raw data according to the current settings. The input path represents the directory in which to store the serialized log files.

```
VQM_800_ERROR StartLogging(CDeviceConnectionInfo connectInfo, string logFolderPath
= "");
```

The logFolderPath must exist, although the default value will result in the use of the path as set by default or by SetLogFileFolderPath. The serialized logs will have a prefix of the Controller's serial number followed by "_data" and a number. The number indicates the order of the files. Raw logs can get very large. To facilitate opening them, the user should set the scan count to be written to each file. See SetLogFileScanCount for details.

The Viewer Application can open a raw log file and play it back. A user might also parse the file and perform analysis on the information.

**StopLogging**

Stops writing to the current raw log file.

```
VQM_800_ERROR StopLogging(CDeviceConnectionInfo connectInfo);
```

This method provides a way for the application to stop raw logging.

### 3.2.12 Gauge States

The gauge state may change as a result of buttons pressed on the front panel of the controller or some other event. An application may want to poll the gauge state periodically to verify that the state has not changed unexpectedly. For example, if your application changes the gauge state to SCAN and then starts calling GetScanData, the application could receive consecutive Scan Timeouts if the Scan button on the controller was pressed and the state has changed to ON (that is the gauge is on but not scaning).

**GetGaugeState**

Reads the current gauge state.

```
VQM_800_ERROR GetGaugeState(out EGaugeState gaugeState, CDeviceConnectionInfo
connectInfo);
```

**SetGaugeState**

Changes the gauge state (for example: off, standby, on, scan) based on the input parameter. Monitors can not perform this command successfully.

```
VQM_800_ERROR SetGaugeState(EGaugeState gaugeState, CDeviceConnectionInfo
connectInfo);
```

Any transition to Off, Standby, On, or Scan is supported. That is, you do not have to step through any intermediate steps to change from Off to Scan. Any necessary intermediate states will be handled. However, an error will be returned if the state change is to one of the External Trigger states.

**StartScan - deprecated**

Starts scanning. This is a legacy call. Consider using SetGaugeState instead.

```
VQM_800_ERROR StartScan(CDeviceConnectionInfo connectInfo);
```

Scanning is initiated and the Controller will start putting raw scan results into the Fetch buffer. These data will be sent to the Data Analysis function for averaging, denoising, and peak finding.

**StopScan - deprecated**

Stops scanning. This is a legacy call. Consider using SetGaugeState instead.

```
VQM_800_ERROR StopScan(CDeviceConnectionInfo connectInfo);
```

Transmit a command to the Controller that will stop the frequency sweep. Power remains applied to the gauge.

**TurnOnGauge - deprecated**

Turns on the power to the gauge sensor. This is a legacy call. Consider the use of SetGaugeState instead.

```
VQM_800_ERROR TurnOnGauge(CDeviceConnectionInfo connectInfo);
```

Apply the current setpoint values to the gauge. The Controller will attempt to prevent damage to the gauge while performing this action and may shut the gauge off if damage is likely.

**TurnOffGauge - deprecated**

Turns off the power to the gauge sensor. This is a legacy call. Consider the use of SetGaugeState instead.

```
VQM_800_ERROR TurnOffGauge(CDeviceConnectionInfo connectInfo);
```

Remove power from the gauge in a controlled fashion (to prevent damage to the gauge).

## 3.3 SendInstruction

We have made every effort to provide a complete API for both Service and Controller interactions. The use of the API is your best protection against unintended Controller and Gauge interactions. However, there may be some activity that you need and that the API does not provide. SendInstruction is your "back door" to the Controller.

SendInstruction provides a method to send a sequence of SCPI commands to the Controller. It does not replace a simple terminal emulator sending SCPI commands and you should not try to implement a terminal emulator using it. The SCPI command set is documented in the 835 VQM Instruction Manual.

***NOTICE:***
***SendInstruction is implemented by the Service and, as such, interacts with the Service. Remember that the Service continues to communicate with the Controller while you are sending the SendInstruction SCPI commands. The interaction may result in unexpected and undesirable results. If you want to send a one-time series of commands to the Controller, it would be better if you disabled the Service, used a terminal emulator to send the commands, and then re-enabled the Service. This sequence will bypass the risks associated with command interaction.***

However, if you need to perform a Controller activity that is not provided by the API and that you need to perform repeatedly, consider using SendInstruction.

```
VQM_800_ERROR SendInstruction(string instr, out string response,
CDeviceConnectionInfo connectInfo);
```

The parameter "instr" is the SCPI command sequence to send to the Controller. The parameter "response" is the returned reply. Note that there may not be a reply to all sequences.

***NOTICE:***
***Because the Service continues to communicate with the Controller while your SCPI command is sent, you will want to concatenate SCPI commands with the ":;" string between the commands when they should not be interrupted. This assures that a Service communication will not interrupt the sequence. For example, while using a terminal emulator, you might send the commands:***

```
INST DOUT
SOUR:DIG:MOD?
```

To find out the current mode for the digital output, one might assume that the equivalent with SendInstruction would be:

```
error = SendInstruction("INST DOUT", out resp, connectInfo);      // wrong!
error = SendInstruction("SOUR:DIG:MOD?", out resp, connectInfo);  // wrong!
```

However, this assumption is incorrect because the Service will perform communications with the Controller between the two SendInstruction calls. There is no way to predict to which logical instrument the second command will be sent.

The correct way to send the sequence (to read the active mode for the digital output) would be:

```
error = SendInstruction("INST DOUT:;SOUR:DIG:MOD?", out resp, connectInfo);
```

Similarly, one could send the command to change the digital output mode to Triggered (the digital output would present a framing pulse for the Analog Output):

```
      error = SendInstruction("INST DOUT:;SOUR:DIG:MOD TRIG", out resp,
connectInfo);
```

Of special insterest is the question mark "?" as a SCPI element. Any SendInstruction call that contains a question mark should end with that question mark.

# 4 LabVIEW VIs

## 4.1 Introduction

Most of the LabVIEW VIs that interface with the Service API are thin wrappers around the underlying API methods. LabVIEW-defined clusters that represent the .NET data structures are provided to allow easy access to the data. There are only a few caveats in using the LabVIEW interface as opposed to the .NET interface.

**Error Handling**

In LabVIEW, the .NET error structures are converted to Error Clusters with an encoded error message. This message encodes the error string and troubleshooting information, if available. This is accomplished by a helper VI that is typically called after each method. Generally, these error clusters can be handled like any other error in LabVIEW.

**Unavailable Functions**

Due to limitations in interpreting some complicated data structures in LabVIEW, there are a few functions that are not available in LabVIEW. In most cases an alternate version with simpler data types is available to get the same data.

**Service Versions**

If a new version of the service is installed, a compatible version of the LabVIEW Interface VIs must also be used. This is due to the fact that the LabVIEW invoke nodes will recognize the change and need to be relinked. Potentially, there could be changes in functionality in newer versions.

## 4.2 Installing the 83x Instrument Driver VIs

The 835 installer copies a zipped package with the LabVIEW VIs, a template and an example to the installation area. This zip archive package needs to be unzipped and the files copied to the indicated locations to develop in LabVIEW uisg these VIs.

1. Navigate to the GP installation folder. Normally, this is C:\Program Files (x86)\GP\VQM but this may vary based on operating system.
2. Navigate to the LabVIEW VIs subdirectory.
3. Unzip the zip archive file called LV 8xx Service API.zip into the LabVIEW VIs subdirectory. This will create a new folder called LV 8xx Service API that contains all of the VIs necessary to ineract with the 8xx Servcie API from LabVIEW.
4. Copy this directory to the inst.lib folder within your LabVIEW 2013 installation directory. Typically this is located in Program Files\National Iinstruments\LabVIEW 2013\user.lib, but may vary depending upon operating sysem or custom install locations. For instance the LV 8xx Service API VIs are 32-bit code which on 64-bit Windows 7 go in the c:\Program Files (x86) folder instead of c:\Program Files.

**Installing the 835 Example File**

1. Navigate to the LabVIEW VIs subdirectory of the GP installation folder.
2. In the LabVIEW VIs subdirectory, there is a VI called 835 API Get Scan Data Example.vi.
3. Copy this file to a new subdirectory called 83x Service Examples in examples within your LabVIEW 2013 installation directory. Normally this would result in C:\Program Files\National Instruments\LabVIEW 2013\examples\83x Service Examples, but it may vary based on operating system or custom install locations.

The VIs will appear on the palette under Instrument IO->Instr Drivers->8xx Svc API. The browse tab on the NI example Finder should also display the 83x APi directory containing an example using these VIs.

## 4.3 LabVIEW 800 Series Service API.lvlib:VI Tree.vi

An organized tree of the driver VIs.


## 4.4 Connections

### 4.4.1 LabVIEW 800 Series Service API.lvlib:8xx Open.vi

Creates a new 8xx Service Reference. Normally this would be used to connect to a device.


**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref**

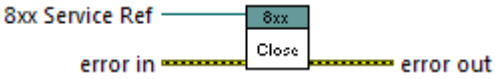
A reference to the CServiceWrapperManager.

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.4.2 LabVIEW 800 Series Service API.lvlib:8xx Close.vi

Closes the 8xx Service Reference.




**8xx Service Ref**

A reference to the CServiceWrapperManager.

**error in**

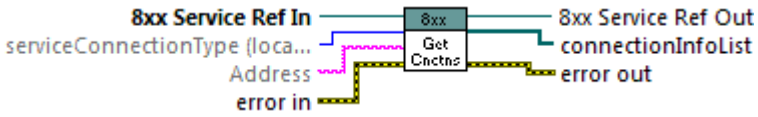
**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.4.3 LabVIEW 800 Series Service API.lvlib:8xx Get Valid Connections.vi

Returns the list of detected controllers.




**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**serviceConnectionType (localhost)**

**Address**

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

**connectionInfoList**

**8xx Connection Info Ref**

### 4.4.4 LabVIEW 800 Series Service API.lvlib:8xx Connect To Device.vi

Connects to a specific controller using the input connection info. Normally this would be returned by a call to 8xx Get Valid Connections.


**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**Connection Info Ref**

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

**Connection Info Ref**

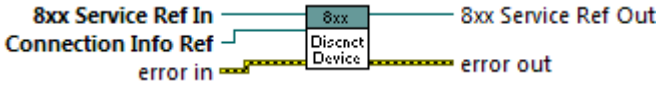
**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

**isMaster**

### 4.4.5 LabVIEW 800 Series Service API.lvlib:8xx Disconnect From Device.vi

Disconnects from and closes the input Connection Info Reference.




**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

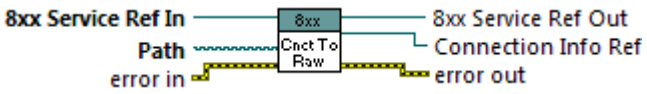
A reference to the CServiceWrapperManager.

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.4.6 LabVIEW 800 Series Service API.lvlib:8xx Connect To Raw Log.vi

Connects to a raw log using the path to the first raw log file in a series of raw log files.




**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**Path**

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

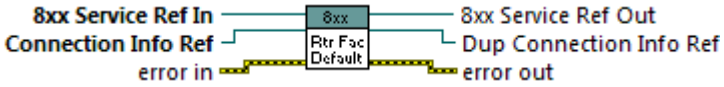
**Connection Info Ref**

## 4.5 Configuration EEPROM

### 4.5.1 Controller Non-Volatile RAM

**LabVIEW 800 Series Service API.lvlib:8xx Restore Factory Defaults.vi**

Restores the factory defaults from the EEPROM. This call can take some time before the controller will be responsive.




**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

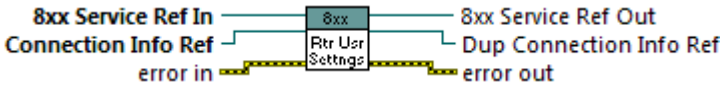
**Dup Connection Info Ref**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

**LabVIEW 800 Series Service API.lvlib:8xx Restore User Settings.vi**

Restores the user settings to the EEPROM. This call can take some time before the controller will be responsive.




**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to

decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

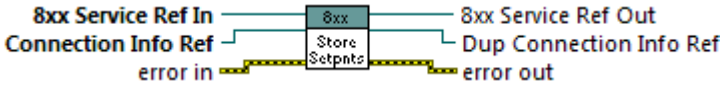
**Dup Connection Info Ref**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

**LabVIEW 800 Series Service API.lvlib:8xx Store Current Setpoints.vi**

Stores the current setpoints to User Settings. These become the values that will be loaded by future calls to Restore User Settings.




**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

**Dup Connection Info Ref**

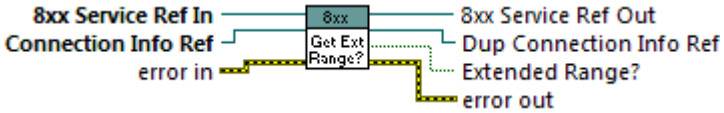
**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.5.2 Controller Operating Parameters

**LabVIEW 800 Series Service API.lvlib:835 Get Ext Range Capabilities.vi**

Determines if the controller has the capability to scan to the extended 300 amu range.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

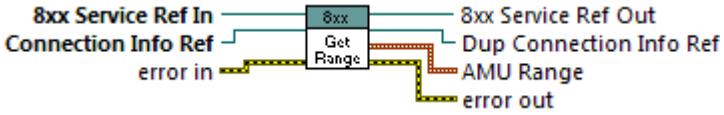
**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

**Extended Range?**

#### LabVIEW 800 Series Service API.lvlib:8xx Get Scan Range.vi

Gets the current setting for the mass scan range that will be swept by the controller for each scan.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

**error out**

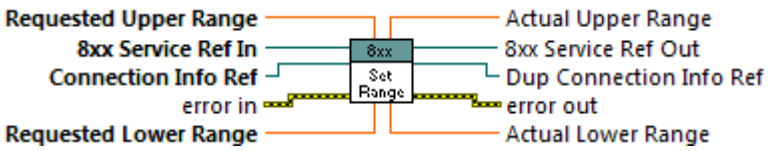
**error out** passes error or warning information out of a VI to be used by other VIs.

**AMU Range**

- **Min**
- **Max**

#### LabVIEW 800 Series Service API.lvlib:8xx Set Scan Range.vi

Returns the most recent Header Data.



**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**Requested Lower Range**

**Requested Upper Range**

**8xx Service Ref Out**

**Dup Connection Info Ref**

**error out**

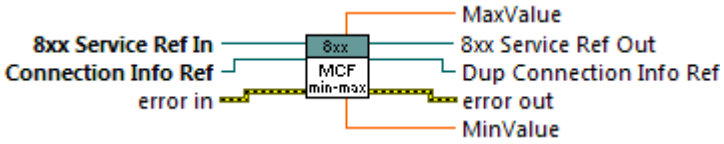
**error out** passes error or warning information out of a VI to be used by other VIs.

**Actual Lower Range**

**Actual Upper Range**

#### LabVIEW 800 Series Service API.lvlib:8xx Get Cal Factor Min Max.vi

Returns the minimum and maximum limits for the Calibration Factor.




**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

**Dup Connection Info Ref**

**error out**

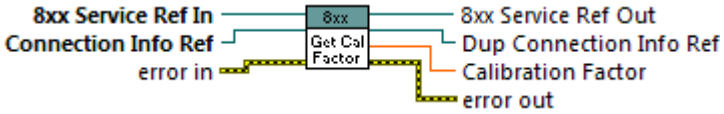
**error out** passes error or warning information out of a VI to be used by other VIs.

**MaxValue**

**MinValue**

#### LabVIEW 800 Series Service API.lvlib:8xx Get Calibration Factor.vi

Returns the most recent value for the Calibration Factor.




**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

**Dup Connection Info Ref**

**Calibration Factor**

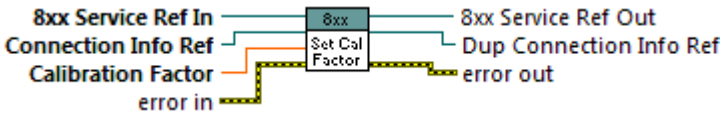
The value of the output calibration factor.

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

#### LabVIEW 800 Series Service API.lvlib:8xx Set Calibration Factor.vi

Sets the value of the Calibration Factor.




**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**Connection Info Ref**

**Calibration Factor**

The value of the calibration factor to be set.

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to

decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

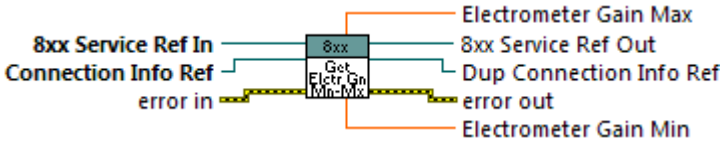
**Dup Connection Info Ref**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

#### LabVIEW 800 Series Service API.lvlib:8xx Get Elctrmtr Gain Min Max.vi

Returns the minimum and maximum allowed value for the Electrometer Gain.




**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

**Dup Connection Info Ref**

**Electrometer Gain Min**

The minimum value allowed for the Electrometer Gain.

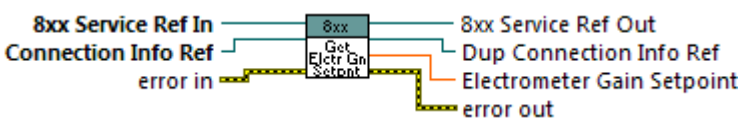
**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

**Electrometer Gain Max**

#### LabVIEW 800 Series Service API.lvlib:8xx Get Elctrmtr Gain Setpoint.vi

Returns the current setpoint for the Electrometer Gain.



**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

**Dup Connection Info Ref**

**Electrometer Gain Setpoint**

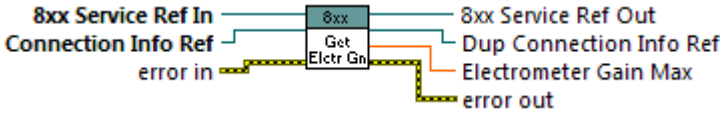
The current setpoint for the Electrometer Gain.

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### LabVIEW 800 Series Service API.lvlib:8xx Get Elctrmtr Gain.vi

Returns the electrometer gain.




**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

**Dup Connection Info Ref**

**Electrometer Gain Max**

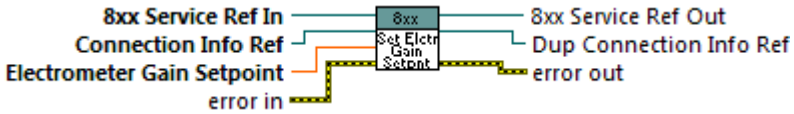
The current value for the Electrometer Gain.

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

**LabVIEW 800 Series Service API.lvlib:8xx Set Elctrmtr Gain Setpoint.vi**

Sets the setpoint for the Electrometer Gain. NOTE: It takes a few seconds until the actual value used by the Controller is read back.




**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**Connection Info Ref**

**Electrometer Gain Setpoint**

The value to be set for the Electrometer Gain setpoint.

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

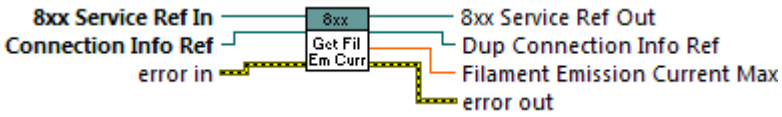
**Dup Connection Info Ref**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

**LabVIEW 800 Series Service API.lvlib:8xx Get Flmnt Emssn Current.vi**

Returns the current Filament Emission Current.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

**Filament Emission Current Max**

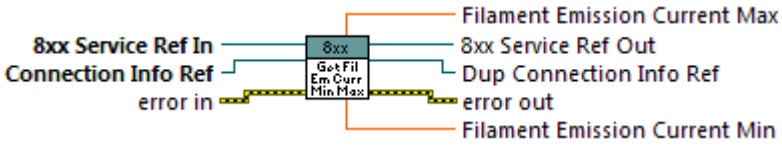
The current value for the Filament Emission Current.

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

#### LabVIEW 800 Series Service API.lvlib:8xx Get Flmnt Emssn Current Min Max.vi

Returns the minimum and maximum allowed values for the Filament Emission Current.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

**Filament Emission Current Min**

The minimum allowed value for the Filament Emission Current.

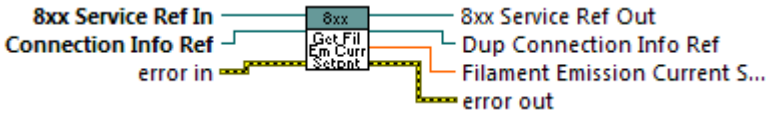
**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

**Filament Emission Current Max**

#### LabVIEW 800 Series Service API.lvlib:8xx Get Flmnt Emssn Setpoint.vi

Returns the current setpoint for the Filament Emission Current.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to

decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

**Filament Emission Current Setpoint**

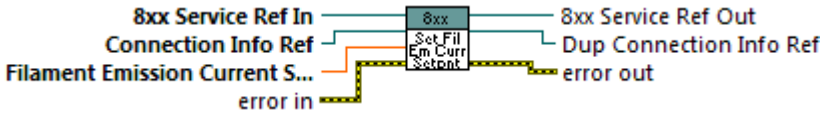
The current setpoint for the Filament Emission Current.

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

#### LabVIEW 800 Series Service API.lvlib:8xx Set Flmnt Emssn Setpoint.vi

Sets the setpoint for the Filament Emission Current.




**8xx Service Ref In**

**Connection Info Ref**

**Filament Emission Current Setpoint**

The value to be set for the Filament Emission setpoint.

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

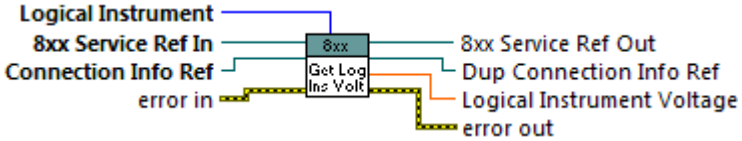
**Dup Connection Info Ref**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

#### LabVIEW 800 Series Service API.lvlib:8xx Get LI Current Voltage.vi

Returns the current voltage for the input logical instrument.




**8xx Service Ref In**

**Connection Info Ref**

**Logical Instrument**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

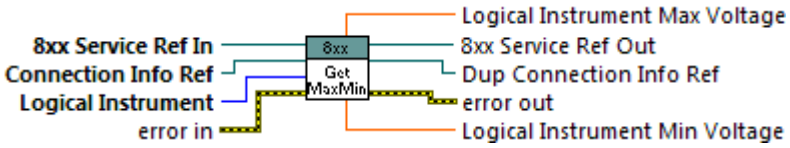
**Logical Instrument Voltage**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

#### LabVIEW 800 Series Service API.lvlib:8xx Get LI Min Max Voltage.vi

Returns the minimum and maximum allowed values for the input logical instrument.




**8xx Service Ref In**

**Connection Info Ref**

**Logical Instrument**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**Logical Instrument Max Voltage**

**8xx Service Ref Out**

**Dup Connection Info Ref**

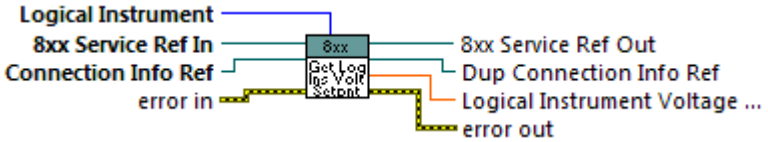
**Logical Instrument Min Voltage**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

#### LabVIEW 800 Series Service API.lvlib:8xx Get LI Voltage Setpoint.vi

Returns the current setpoint for the input logical instrument.



**Logical Instrument**

**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

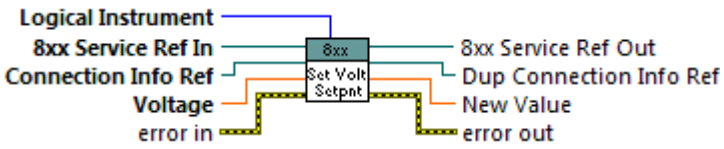
**Logical Instrument Voltage Setpoint**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

#### LabVIEW 800 Series Service API.lvlib:8xx Set LI Voltage Setpoint.vi

Sets the voltage setpoint for the input logical instrument.




**Logical Instrument**

**8xx Service Ref In**

**Connection Info Ref**

**Voltage**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

**error out**

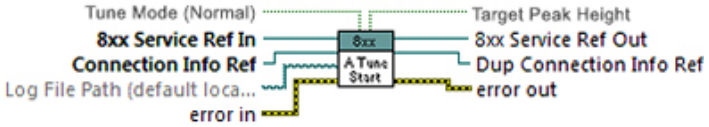
**error out** passes error or warning information out of a VI to be used by other VIs.

**New Value**

### 4.5.3 Auto Tune

**LabVIEW 800 Series Service API.lvlib:8xx Auto Tune Start.vi**

Starts the autotune process.




**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**Log File Path (default location)**

**Tune Mode (Normal)**

Specifies which Auto Tune mode – Normal, Detailed, Express, or Electron Multiplier only.

**Target Peak Height**

Longer EM life 8na, Normal 20na, High Sensitivity 50na.

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

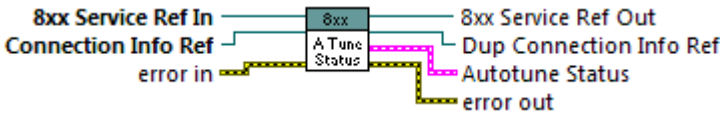
**Dup Connection Info Ref**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

**LabVIEW 800 Series Service API.lvlib:8xx Auto Tune Get Status.vi**

Queries the auto tune status.




**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

**Dup Connection Info Ref**

**error out**

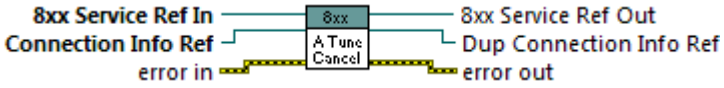
**error out** passes error or warning information out of a VI to be used by other VIs.

**Autotune Status**

- **Error?**
- **status**
- **Percent Done**
- **EM Bias**
- **Repeller**
- **Exit Plate**
- **Shield**
- **RF Amp PP**
- **Filament Emission Current**

### LabVIEW 800 Series Service API.lvlib:8xx Auto Tune Cancel.vi

Cancels the autotune process.




**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

**Dup Connection Info Ref**

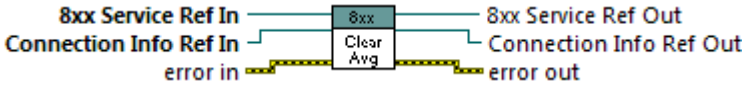
**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.5.4 Averaging

**LabVIEW 800 Series Service API.lvlib:8xx Clear Avg History.vi**

Clears the average history of all previous samples.




**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**Connection Info Ref In**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

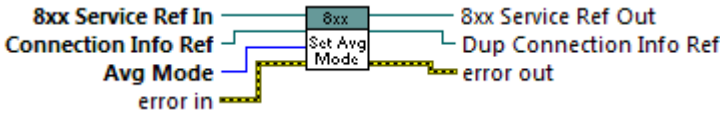
**Connection Info Ref Out**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

**LabVIEW 800 Series Service API.lvlib:8xx Set Avg Mode.vi**

Sets the average mode for the averager in data analysis.




**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**Connection Info Ref**

**Avg Mode**

The averaging mode that will be used.

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to

decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

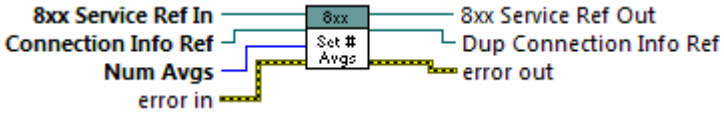
**Dup Connection Info Ref**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

**LabVIEW 800 Series Service API.lvlib:8xx Set Num Avgs.vi**

Sets the number of averages used by the averager to produce analyzed data.




**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**Connection Info Ref**

**Num Avgs**

The number of averages used by the averager to calculate the averaged pressure and scan data.

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

**Dup Connection Info Ref**

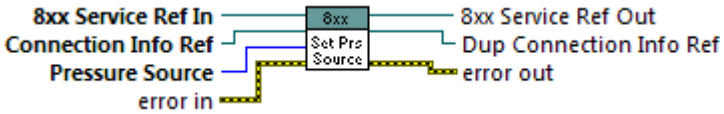
**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.5.5 Total Pressure

**LabVIEW 800 Series Service API.lvlib:8xx Set Pressure Source.vi**

Sets the pressure source to be used for analyzed data.



**8xx Service Ref In**

A reference to the CServiceWrapperManager.

**Connection Info Ref**

**Pressure Source**

Name of the Total Pressure Source. Sets the pressure source used by data averaging to compute the average total pressure. The name must be on the list of pressure sources returned from 8xx Get Total Pressure Gauges.vi.

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

A reference to the CServiceWrapperManager.

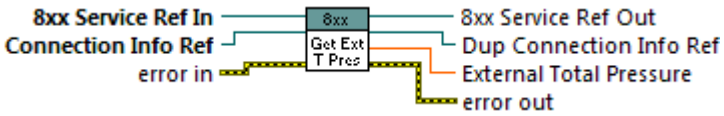
**Dup Connection Info Ref**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### LabVIEW 800 Series Service API.lvlib:8xx Get External Total Pressure.vi

Gets the current value from the external total pressure sensor. Typically, this is a 390 pressure sensor, although the total pressure may also be sourced by a variety of analog input total pressure sensors.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

**External Total Pressure**

The external total pressure.

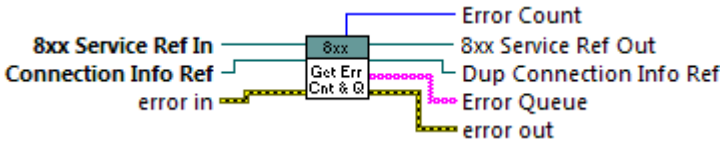
**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

## 4.6 Errors

### 4.6.1 LabVIEW 800 Series Service API.lvlib:8xx Get Error Count and Queue.vi

Gets the errors from the controller error queue. The errors will be cleared from the queue after they have been returned.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

**Error Count**

The number of errors on the controller.

**Error Queue**

A list of all the errors on the controller.

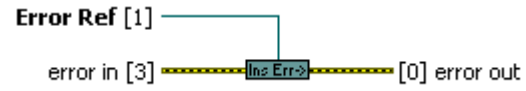
**String**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.6.2 LabVIEW 800 Series Service API.lvlib:8xx Insert Error.vi

Converts an error returned by the .NET assembly into a LabVIEW error Cluster.




**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**Error Ref**

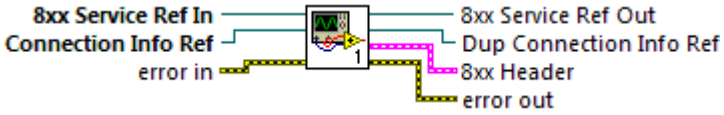
**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

## 4.7 Header Data, Scan Data, Analysis Results

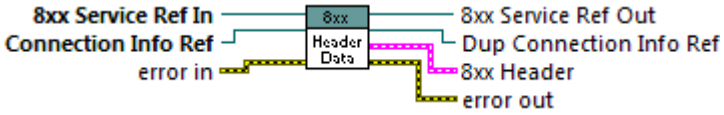
### 4.7.1 LabVIEW 800 Series Service API.lvlib:8xx Get Header Data.vi

This is the polymorphic version for Get Header Data. The only supported controller is the 835. For details of the returned data, See 4.7.2(LabVIEW 800 Series Service API.lvlib:835 Get Header Data.vi).




### 4.7.2 LabVIEW 800 Series Service API.lvlib:835 Get Header Data.vi

Returns the most recent Header Data.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

**8xx Header**

Scan header information.

- **Sensor Values**
  - **Filament Emission Current**
  - **Filament Bias Voltage**
  - **Repeller Voltage**
  - **Entry Plate Voltage**
  - **Pressure Plate Voltage**
  - **Cups Voltage**
  - **Transition Voltage**
  - **Exit Plate Voltage**
  - **Electron Multipler Shield Voltage**

- Electron Multipler Bias Voltage
- RF Amp Voltage
- Mass Cal Factor
- Electrometer Gain
- Status
  - STB
  - SESR
  - QUES:CONT
  - QUES:ION
  - QUES:MSEP
  - QUES:DET
  - QUES:ESIG
  - QUES:ETPR1
  - QUES:ETPR2
  - QUES:ETPR3
  - OPER:CONT
  - OPER:ION
  - OPER:MSEP
  - OPER:DET
  - OPER:ESIG
  - OPER:ETPR
- Scan Params
  - Timestamp
  - OWrt Counter
  - Units
  - TSource
  - OTrg Counter
  - NSamples
  - Begin AMU
  - End AMU
  - Filament Power

**Pressure Params**

**Pressure TS**

**Pressure Units**

**Pressure**

**Ext Signal Params**

**Ext Signal TS**

**Ext Signal Units**

**Ext Signal**

**Device Specific**

**Version**

**Date**

**Time**

**Serial Number**

**HW Rev**

**FW Rev**

**Data length**

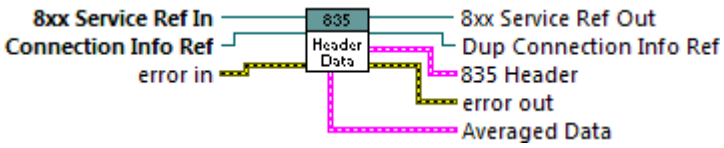
**Has Errors?**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.7.3 LabVIEW 800 Series Service API.lvlib:835 Get Header and Avg Data.vi

Returns the most recent Header Data and most recent average data.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

**835 Header**

Scan header information.

**Sensor Values**

**Filament Emission Current**

**Filament Bias Voltage**

**Repeller Voltage**

**Entry Plate Voltage**

**Pressure Plate Voltage**

**Cups Voltage**

**Transition Voltage**

**Exit Plate Voltage**

**Electron Multipler Shield Voltage**

**Electron Multipler Bias Voltage**

Electron Multiplier Voltage. Range is -500 to -1500V.

**RF Amp Voltage**

**Mass Cal Factor**

**Electrometer Gain**

**Status**

**STB**

**SESR**

**QUES:CONT**

**QUES:ION**

**QUES:MSEP**

**QUES:DET**

**QUES:ESIG**

**QUES:ETPR1**

- QUES:ETPR2
- QUES:ETPR3
- OPER:CONT
- OPER:ION
- OPER:MSEP
- OPER:DET
- OPER:ESIG
- OPER:ETPR

**Scan Params**

- Timestamp
- OWrt Counter
- Units
- TSource
- OTrg Counter
- NSamples
- Begin AMU
- End AMU
- Filament Power

**Pressure Params**

- Pressure TS
- Pressure Units
- Pressure

**Ext Signal Params**

- Ext Signal TS
- Ext Signal Units
- Ext Signal

**Device Specific**

- Version
- Date
- Time
- Serial Number

**HW Rev**

**FW Rev**

**Data length**

**Has Errors?**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

**Averaged Data**

The averaged data.

**Avg Mode**

Averaging mode used by the averager.

**Source**

Pressure Source (typically set by user).

**Inst Total Pres**

Instantaneous total pressure value.

**Avg Total Pres**

Averaged Total Pressure.

**Buffers Averaged**

Number of samples collected so far.

**Time Stamp**

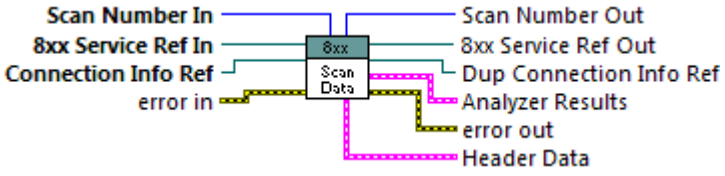
**Ctrlr Time Stamp**

**Averages Valid?**

### 4.7.4 LabVIEW 800 Series Service API.lvlib:8xx Get Scan Data.vi

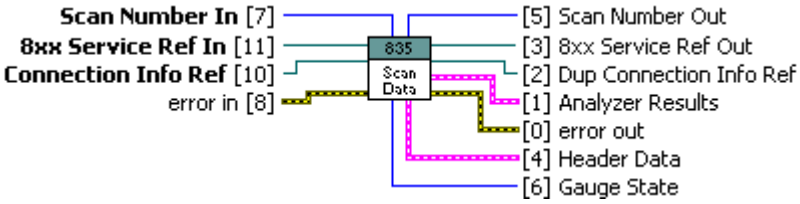
This is the polymorphic version of Get Scan Data. The only supported controller at this time is the 835. See the description in 4.7.5(LabVIEW 800 Series Service API.lvlib:835 Get Scan Data.vi) for details for the 835.

Gets the current scan data. This consists of Header Data, Analyzer Results, and a Scan Number. The Scan Number may be used to detect duplicate or missed scans.



### 4.7.5 LabVIEW 800 Series Service API.lvlib:835 Get Scan Data.vi

Gets the current scan data. This consists of Header Data, Analyzer Results, and a Scan Number. The Scan Number may be used to detect duplicate or missed scans.




**8xx Service Ref In**

**Connection Info Ref**

**Scan Number In**

The scan number of the requested data. This can be compared with the actual scan number retrieved to detect missed scans.

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**Scan Number Out**

The scan number of the actual data. This can be compared with the requested scan number to detect missed scans.

**Header Data**

- **Sensor Values**
  - **Filament Emission Current**
  - **Filament Bias Voltage**
  - **Repeller Voltage**
  - **Entry Plate Voltage**
  - **Pressure Plate Voltage**
  - **Cups Voltage**
  - **Transition Voltage**
  - **Exit Plate Voltage**
  - **Electron Multipler Shield Voltage**
  - **Electron Multipler Bias Voltage**

    Electron Multiplier Voltage. Range is -500 to -1500V.
  - **RF Amp Voltage**

- Mass Cal Factor
- Electrometer Gain

Status

- STB
- SESR
- QUES:CONT
- QUES:ION
- QUES:MSEP
- QUES:DET
- QUES:ESIG
- QUES:ETPR1
- QUES:ETPR2
- QUES:ETPR3
- OPER:CONT
- OPER:ION
- OPER:MSEP
- OPER:DET
- OPER:ESIG
- OPER:ETPR

Scan Params

- Timestamp
- OWrt Counter
- Units
- TSource
- OTrg Counter
- NSamples
- Begin AMU
- End AMU
- Filament Power

Pressure Params

- Pressure TS

Pressure Units

Pressure

Ext Signal Params

Ext Signal TS

Ext Signal Units

Ext Signal

Device Specific

Version

Date

Time

Serial Number

HW Rev

FW Rev

Data length

Has Errors?

8xx Service Ref Out

Dup Connection Info Ref

Analyzer Results

The analyzed scan data results.

Averaged Data

The averaged data.

Avg Mode

Averaging mode used by the averager.

Source

Pressure Source (typically set by user).

Inst Total Pres

Instantaneous total pressure value.

Avg Total Pres

Averaged Total Pressure.

Buffers Averaged

Number of samples collected so far.

Time Stamp

Ctrlr Time Stamp

Averages Valid?

**Averaged ADC Output**

Averaged Counts as computed by the averager.

**Averages Valid**

Data is typically only invalid when Accumulator averaging is not yet ready or if the averager has just been reset.

**Scaled RMS Noise**

Scaled RMS background noise.

**Denoised Curve**

The raw curve data.

**Mass_Axis**

**Mass**

Just the mass axis as computed by the frequency to mass calculation.

**Denoised Raw**

**Intensity**

Denoised intensity curve, analogous to Counts. Typically displayed as a curve as in the tuning curve.

**Histogram Data**

The histogram data.

**Peak Area**

**Numeric**

The Peak Area for each mass peak.

**GOF**

**Numeric**

The Goodness Of Fit for each mass peak.

**FWHM**

**Numeric**

The Full Width Half Max of each mass peak.

**Resolution**

**Numeric**

The Resolution for each mass peak.

**Amplitude**

**Numeric**

The amplitude of each mass peak.

**Center**

**Numeric**

The Center value for each mass peak.

**Fitted Gases**

**G&R Pressure**

Gas Fractions as computed by gasfit.

**Residuals**

Residual Fractions as computed by gasfit.

**G&R Pressure**

**Valid?**

**Total Area**

**Noise Baseline**

**High Mass Area**

**Averaged Noise Data**

Averaged Counts as computed by the averager.

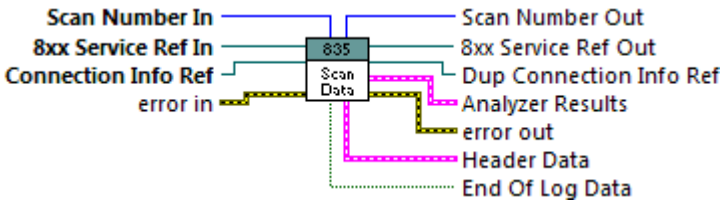
**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

**Gauge State**

### 4.7.6 LabVIEW 800 Series Service API.lvlib:835 Get Scan Data From Log.vi

Gets the scan data from the log This consists of Header Data, Analyzer Results, and a Scan Number. The Scan Number may be used to detect duplicate or missed scans.




**8xx Service Ref In**

**Connection Info Ref**

**Scan Number In**

The scan number of the requested data. This can be compared with the actual scan number retrieved to detect missed scans.

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**Scan Number Out**

The scan number of the actual data. This can be compared with the requested scan number to detect missed scans.

**Header Data**

- **Sensor Values**
  - **Filament Emission Current**
  - **Filament Bias Voltage**
  - **Repeller Voltage**
  - **Entry Plate Voltage**
  - **Pressure Plate Voltage**
  - **Cups Voltage**
  - **Transition Voltage**
  - **Exit Plate Voltage**
  - **Electron Multipler Shield Voltage**
  - **Electron Multipler Bias Voltage**

    Electron Multiplier Voltage. Range is -500 to -1500V.
  - **RF Amp Voltage**
  - **Mass Cal Factor**
  - **Electrometer Gain**
- **Status**

- U8 STB
- U8 SESR
- U16 QUES:CONT
- U16 QUES:ION
- U16 QUES:MSEP
- U16 QUES:DET
- U16 QUES:ESIG
- U16 QUES:ETPR1
- U16 QUES:ETPR2
- U16 QUES:ETPR3
- U16 OPER:CONT
- U16 OPER:ION
- U16 OPER:MSEP
- U16 OPER:DET
- U16 OPER:ESIG
- U16 OPER:ETPR

Scan Params

- Timestamp
- U32 OWrt Counter
- abc Units
- abc TSource
- U32 OTrg Counter
- I32 NSamples
- DBL Begin AMU
- DBL End AMU
- DBL Filament Power

Pressure Params

- Pressure TS
- abc Pressure Units
- DBL Pressure

Ext Signal Params

**Ext Signal TS**

**Ext Signal Units**

**Ext Signal**

**Device Specific**

**Version**

**Date**

**Time**

**Serial Number**

**HW Rev**

**FW Rev**

**Data length**

**Has Errors?**

**8xx Service Ref Out**

**Dup Connection Info Ref**

**Analyzer Results**

The analyzed scan data results.

**Averaged Data**

The averaged data.

**Avg Mode**

Averaging mode used by the averager.

**Source**

Pressure Source (typically set by user).

**Inst Total Pres**

Instantaneous total pressure value.

**Avg Total Pres**

Averaged Total Pressure.

**Buffers Averaged**

Number of samples collected so far.

**Time Stamp**

**Ctrlr Time Stamp**

**Averages Valid?**

**Averaged ADC Output**

Averaged Counts as computed by the averager.

**Averages Valid**

Data is typically only invalid when Accumulator averaging is not yet ready or if the averager has just been reset.

**Scaled RMS Noise**

Scaled RMS background noise.

**Denoised Curve**

The raw curve data.

**Mass_Axis**

**Mass**

Just the mass axis as computed by the frequency to mass calculation.

**Denoised Raw**

**Intensity**

Denoised intensity curve, analogous to Counts. Typically displayed as a curve as in the tuning curve.

**Histogram Data**

The histogram data.

**Peak Area**

**Numeric**

The Peak Area for each mass peak.

**GOF**

**Numeric**

The Goodness Of Fit for each mass peak.

**FWHM**

**Numeric**

The Full Width Half Max of each mass peak.

**Resolution**

**Numeric**

The Resolution for each mass peak.

**Amplitude**

**Numeric**

The amplitude of each mass peak.

**Center**

**Numeric**

The Center value for each mass peak.

**Fitted Gases**

**G&R Pressure**

Gas Fractions as computed by gasfit.

**Residuals**

Residual Fractions as computed by gasfit.

**G&R Pressure**

**Valid?**

**Total Area**

**Noise Baseline**

**High Mass Area**

**Averaged Noise Data**

Averaged Counts as computed by the averager.

**error out**

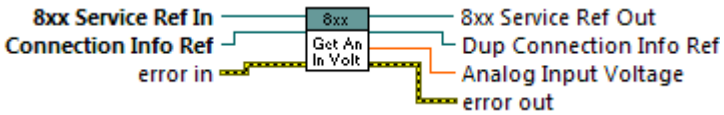
**error out** passes error or warning information out of a VI to be used by other VIs.

**End Of Log Data**

## 4.8 Analog and Digital I/O

### 4.8.1 LabVIEW 800 Series Service API.lvlib:8xx Get Analog Input Voltage.vi

Returns the current value of the external signal voltage. Typically, this will represent an analog external pressure sensor. Use this API if the Analog Input is not connected to a Total Pressure device. If the Analog Input is connected to a Total Pressure device, set the Total Pressure Source appropriately and use the Get External Total Pressure API instead.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

**Analog Input Voltage**

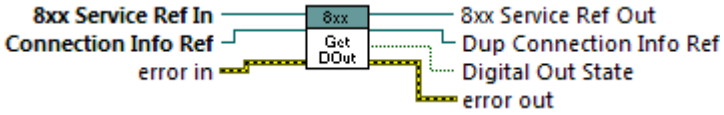
The external analog input voltage.

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.8.2 LabVIEW 800 Series Service API.lvlib:835 Get Digital Out.vi

Gets the value of the Digital Output on the controller.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

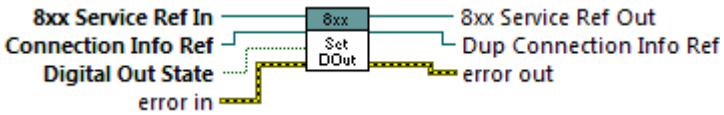
**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

**Digital Out State**

### 4.8.3 LabVIEW 800 Series Service API.lvlib:835 Set Digital Out.vi

Sets the value of the Digital Output on the controller.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**Digital Out State**

**8xx Service Ref Out**

**Dup Connection Info Ref**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

## 4.9 Gas-Fitting Definitions

These methods are most useful for identifying the gases (by name) that are returned in the Get Scan Data results. The order of the gases is the same for both.

### 4.9.1 LabVIEW 800 Series Service API.lvlib:8xx Get Combo File Data Direct.vi

Outputs the current combo data, which is the data used for the gas fitting algorithm. The Gas Names are provided separately as a convenience, but it is the same name as in the array of combo data.


**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

**Gas Names**

- **GasName**

**Combo Data**

- **ComboData**
  - **Gas Name**
  - **Fitted Amu Values**
    - **Numeric**
  - **Fitted Gases**
    - **Numeric**

Residual Amu Values

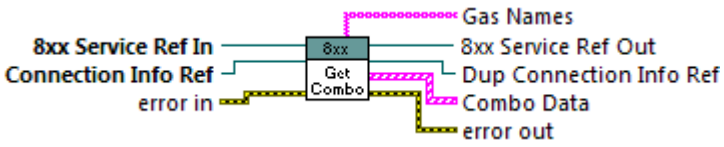
Numeric

Residuals

Numeric

### 4.9.2 LabVIEW 800 Series Service API.lvlib:8xx Get Combo File Data.vi

Outputs the current combo data, which is the data used for the gas fitting algorithm. The Gas Names are provided separately as a convenience, but it is the same name as in the array of combo data. Use this version when using a tcp (remote) connection to ensure to get the data being used to gas fit.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

**Gas Names**

GasName

**Combo Data**

ComboData

Gas Name

Fitted Amu Values

Numeric

Fitted Gases

Numeric

Residual Amu Values

Numeric

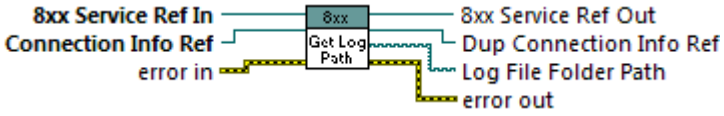
**Residuals**

**Numeric**

## 4.10 Logging

### 4.10.1 LabVIEW 800 Series Service API.lvlib:8xx Get Log File Folder Path.vi

Gets currently configured path to the log directory.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

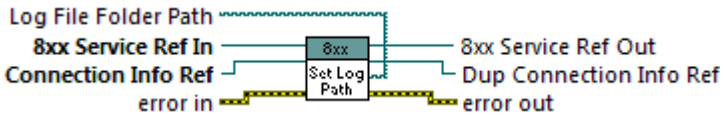
**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

**Log File Folder Path**

### 4.10.2 LabVIEW 800 Series Service API.lvlib:8xx Set Log File Folder Path.vi

Sets the currently configured path to the log directory.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**Log File Folder Path**

**8xx Service Ref Out**

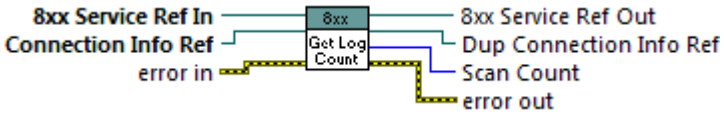
**Dup Connection Info Ref**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.10.3 LabVIEW 800 Series Service API.lvlib:8xx Get Log File Scan Count.vi

Gets the current number of scans per serialized log file.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

**Scan Count**

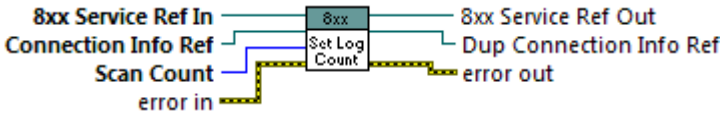
The number of scans in a log file before it will roll to a new serialized log file.

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.10.4 LabVIEW 800 Series Service API.lvlib:8xx Set Log File Scan Count.vi

Sets the current number of scans per serialized log file.




**8xx Service Ref In**

**Connection Info Ref**

**Scan Count**

The set number of scans in a log file before it will roll to a new serialized log file.

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.10.5 LabVIEW 800 Series Service API.lvlib:8xx Start Logging.vi

Starts logging raw data according to the current settings. The input path will represent the first log file in the serialized log files.


**8xx Service Ref In**

**Connection Info Ref**

**Log Path**

Path of the file to log to.

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

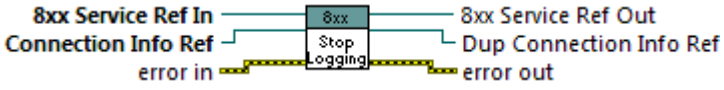
**Dup Connection Info Ref**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.10.6 LabVIEW 800 Series Service API.lvlib:8xx Stop Logging.vi

Stops writing to the current raw log file.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

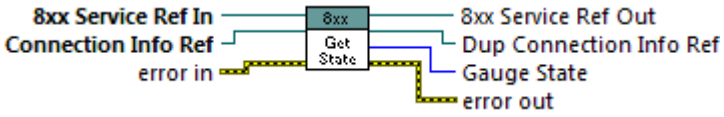
**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

## 4.11 Gauge States

### 4.11.1 LabVIEW 800 Series Service API.lvlib:8xx Get Gauge State.vi

Gets the current gauge state – Off, Standby, On, or Scan..




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

**error out**

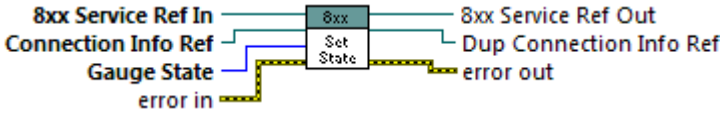
**error out** passes error or warning information out of a VI to be used by other VIs.

**Gauge State**

Off, Standby, On, or Scan

### 4.11.2 LabVIEW 800 Series Service API.lvlib:8xx Set Gauge State.vi

Sets the current state of the sensor gauge to the input state. This is the preferred method of changing the state of the gauge.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**Gauge State**

**8xx Service Ref Out**

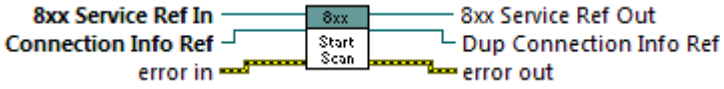
**Dup Connection Info Ref**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.11.3 LabVIEW 800 Series Service API.lvlib:8xx Start Scan.vi

Starts continuous scanning.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

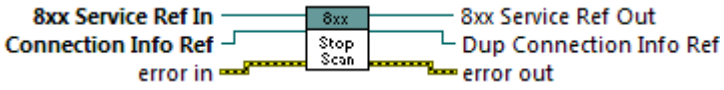
**Dup Connection Info Ref**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.11.4 LabVIEW 800 Series Service API.lvlib:8xx Stop Scan.vi

Stops continuous scanning.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

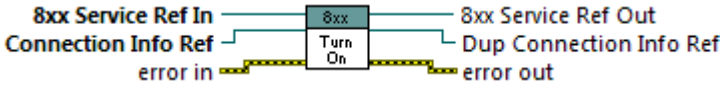
**Dup Connection Info Ref**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.11.5 LabVIEW 800 Series Service API.lvlib:8xx Turn On Gauge.vi

Turns on the power to the gauge sensor but does not initiate scanning.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

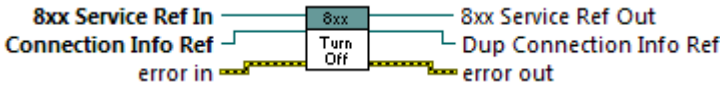
**Dup Connection Info Ref**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.11.6 LabVIEW 800 Series Service API.lvlib:8xx Turn Off Gauge.vi

Turns off the power to the gauge sensor.




**8xx Service Ref In**

**Connection Info Ref**

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref Out**

**Dup Connection Info Ref**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

## 4.12 Helpers

### 4.12.1 LabVIEW 800 Series Service API.lvlib:835 Extract Header Data.vi

Utility VI to build the Header Data from the Results Reference returned by the service.


**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**835 Header Data Ref**

**835 Header**

- **Sensor Values**
  - **Filament Emission Current**
  - **Filament Bias Voltage**
  - **Repeller Voltage**
  - **Entry Plate Voltage**
  - **Pressure Plate Voltage**
  - **Cups Voltage**
  - **Transition Voltage**
  - **Exit Plate Voltage**
  - **Electron Multipler Shield Voltage**
  - **Electron Multipler Bias Voltage**

    Electron Multiplier Voltage. Range is -500 to -1500V.
  - **RF Amp Voltage**
  - **Mass Cal Factor**
  - **Electrometer Gain**
- **Status**
  - **STB**
  - **SESR**

- U16 QUES:CONT
- U16 QUES:ION
- U16 QUES:MSEP
- U16 QUES:DET
- U16 QUES:ESIG
- U16 QUES:ETPR1
- U16 QUES:ETPR2
- U16 QUES:ETPR3
- U16 OPER:CONT
- U16 OPER:ION
- U16 OPER:MSEP
- U16 OPER:DET
- U16 OPER:ESIG
- U16 OPER:ETPR

Scan Params

- Timestamp
- U32 OWrt Counter
- abc Units
- abc TSource
- U32 OTrg Counter
- I32 NSamples
- DBL Begin AMU
- DBL End AMU
- DBL Filament Power

Pressure Params

- Pressure TS
- abc Pressure Units
- DBL Pressure

Ext Signal Params

- Ext Signal TS
- abc Ext Signal Units

Ext Signal

Device Specific

Version

Date

Time

Serial Number

HW Rev

FW Rev

Data length

Has Errors?

error out

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**status**

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.

**code**

**code** is the error or warning code.

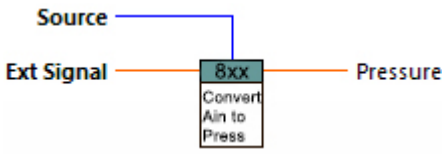
**source**

**source** describes the origin of the error or warning.

### 4.12.2 LabVIEW 800 Series Service API.lvlib:8xx Convert Analog In to Pressure.vi

Simple VI to just convert external signal (analog) to pressure value. This same feature is available by setting the total pressure source to the correct Analog Input Source and calling Get External Total Pressure.

Copyright: MKS



**Source**

The pressure source name. This must match a name in the list returned from the 8xx Get Total Pressure Gauges.vi.

**Ext Signal**

The external signal.

**Pressure**

The value of the pressure.

### 4.12.3 LabVIEW 800 Series Service API.lvlib:8xx Convert Date-Time to TS.vi

Utility VI to convert returned Date and Time strings to a LabVIEW timestamp datatype.


**CurveDate**

The Curve Date as returned by the header data.

**CurveTime**

The Curve Time as returned by the header data.

**Time Stamp**

The converted time stamp.

---

### 4.12.4 LabVIEW 800 Series Service API.lvlib:8xx Convert LV TS to Service TS.vi

Converts a LabVIEW timestamp into the timestamp used by the service.


**Time Stamp**

The LabVIEW timestamp.

**Service Timestamp**

The service timestamp.

### 4.12.5 LabVIEW 800 Series Service API.lvlib:8xx Convert Service TS to LV TS.vi

Converts the timestamp used by the service into a LabVIEW timestamp.

**Service Timestamp**

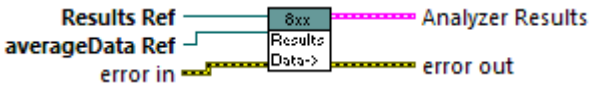
The service timestamp.

**Time Stamp**

The LabVIEW timestamp.

### 4.12.6 8xx Extract Analyzed Results Data.vi

Utility VI to build the Analyzer Results from the Results Reference returned by the service.




**Results Ref**

A reference to the results.

**averageData Ref**

A reference to the averaged data.

**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

**Analyzer Results**

The analyzer results.

**Averaged Data**

The averaged data.

**Avg Mode**

Averaging mode used by the averager.

**Source**

Pressure Source (typically set by user).

**Inst Total Pres**

Instantaneous total pressure value.

**Avg Total Pres**

Averaged Total Pressure.

**Buffers Averaged**

Number of samples collected so far.

**Time Stamp**

**Ctrlr Time Stamp**

**Averages Valid?**

**Averaged ADC Output**

Averaged Counts as computed by the averager.

**Averages Valid**

Data is typically only invalid when Accumulator averaging is not yet ready or if the averager has just been reset.

**Scaled RMS Noise**

Scaled RMS background noise.

**Denoised Curve**

The raw curve data.

**Mass_Axis**

**Mass**

Just the mass axis as computed by the frequency to mass calculation.

**Denoised Raw**

**Intensity**

Denoised intensity curve, analogous to Counts. Typically displayed as a curve as in the tuning curve.

**Histogram Data**

The histogram data.

**Peak Area**

**Numeric**

The Peak Area for each mass peak.

**GOF**

**Numeric**

The Goodness Of Fit for each mass peak.

**FWHM**

**Numeric**

The Full Width Half Max of each mass peak.

**Resolution**

**Numeric**

The Resolution for each mass peak.

**Amplitude**

**Numeric**

The amplitude of each mass peak.

**Center**

**Numeric**

The Center value for each mass peak.

**Fitted Gases**

**G&R Pressure**

Gas Fractions as computed by gasfit.

**Residuals**

Residual Fractions as computed by gasfit.

**G&R Pressure**

**Valid?**

**Total Area**

**Noise Baseline**

**High Mass Area**

**Averaged Noise Data**

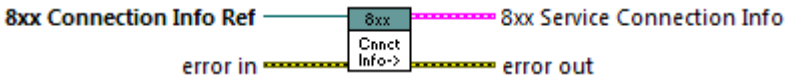
Averaged Counts as computed by the averager.

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.12.7 LabVIEW 800 Series Service API.lvlib:8xx Extract Connection Info.vi

Utility VI to build the Header Data from the Results Reference returned by the service.




**error in**

**error in** can accept error information wired from VIs previously called. Use this information to

decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Connection Info Ref**

**error out**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**status**

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.

**code**

**code** is the error or warning code.

**source**

**source** describes the origin of the error or warning.

**8xx Service Connection Info**

**Availability**

**Port**

The COM port.

**DeviceType**

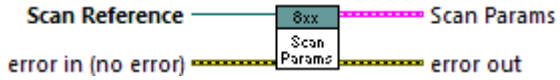
**SN**

Serial number of the connected controller.

### 4.12.8 LabVIEW 800 Series Service API.lvlib:8xx Extract Header Data.vi


### 4.12.9 8xx Extract Noise Parameters.vi

Extracts the scan parameters from the .NET ADC Output structure.



**Scan Reference**

**error in (no error)**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

**Scan Params**

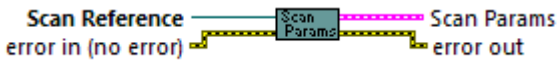
- **Timestamp**
- **OWrt Counter**
- **Units**
- **TSource**
- **OTrg Counter**
- **NSamples**
- **Begin AMU**
- **End AMU**
- **Filament Power**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.12.10 LabVIEW 800 Series Service API.lvlib:8xx Extract Scan Parameters.vi

Extracts the scan parameters from the .NET ADC Output structure.




**Scan Reference**

**error in (no error)**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

**Scan Params**

- **Timestamp**
- **OWrt Counter**

**Units**

**TSource**

**OTrg Counter**

**NSamples**

**Begin AMU**

**End AMU**

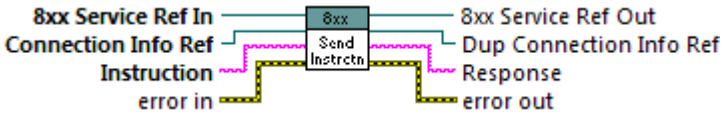
**Filament Power**

**error out**

**error out** passes error or warning information out of a VI to be used by other VIs.

### 4.12.11 LabVIEW 800 Series Service API.lvlib:8xx Send Instruction.vi

Sends an arbitrary instruction to the controller. See the VQM System manual (p/n 835000) for SCPI commands.




**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**8xx Service Ref In**

**Connection Info Ref**

**Instruction**

**error out**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**status**

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.

**code**

**code** is the error or warning code.

**source**

**source** describes the origin of the error or warning.

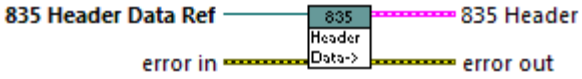
**8xx Service Ref Out**

**Dup Connection Info Ref**

**Response**

### 4.12.12 LabVIEW 800 Series Service API.lvlib:835 Extract Header Data.vi

Utility VI to build the Header Data from the Results Reference returned by the service.




**error in**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**835 Header Data Ref**

**835 Header**

**Sensor Values**

**Filament Emission Current**

**Filament Bias Voltage**

**Repeller Voltage**

**Entry Plate Voltage**

**Pressure Plate Voltage**

**Cups Voltage**

**Transition Voltage**

**Exit Plate Voltage**

**Electron Multipler Shield Voltage**

**Electron Multipler Bias Voltage**

Electron Multiplier Voltage. Range is -500 to -1500V.

**RF Amp Voltage**

**Mass Cal Factor**

**Electrometer Gain**

**Status**

**STB**

**SESR**

**QUES:CONT**

- U16 QUES:ION
- U16 QUES:MSEP
- U16 QUES:DET
- U16 QUES:ESIG
- U16 QUES:ETPR1
- U16 QUES:ETPR2
- U16 QUES:ETPR3
- U16 OPER:CONT
- U16 OPER:ION
- U16 OPER:MSEP
- U16 OPER:DET
- U16 OPER:ESIG
- U16 OPER:ETPR

Scan Params

- Timestamp
- U32 OWrt Counter
- abc Units
- abc TSource
- U32 OTrg Counter
- I32 NSamples
- DBL Begin AMU
- DBL End AMU
- DBL Filament Power

Pressure Params

- Pressure TS
- abc Pressure Units
- DBL Pressure

Ext Signal Params

- Ext Signal TS
- abc Ext Signal Units
- DBL Ext Signal

**Device Specific**

**Version**

**Date**

**Time**

**Serial Number**

**HW Rev**

**FW Rev**

**Data length**

**Has Errors?**

**error out**

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs.

**status**

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred.

**code**

**code** is the error or warning code.

**source**

**source** describes the origin of the error or warning.

**Time Stamp**

The converted time stamp.

# 5 C# Programming

A sample C# application using the APIs is included in the <install directory>[3]\API Files\Net\Samples. The csproj file was created using Visual Studio 2010.

# 6 C++ Programming

Use Visual Studio to create references to CSServiceWrapper.dll and Utilities.dll in <install directory>[3]\API files\NET\Assemblies.

For errors returned from the API that advise checking the error log for details of failures, look at the appropriate 800 Series error log. The default location for these error logs is in <public documents>[4]\VQM\ErrorLogs. There is a separate error log for each of the possible four services running.

For C++ programming, the necessary files are located in subfolders in <install directory>[3]\API files\C++. These folders also contain some template projects as examples. The CLIWrapperDLL.dll is used for debugging. To set up a project to use the CLIWrapperDLL make the following changes:

1. Under C/C++ General settings, for the property "Additional Include Directories", add <install directory>[3]VQM\835\API files\C++\Libs.
2. Under Linker General Settings, for "Addiitonal Library Directories", add <install directory>[3]\API files\C++\Libs
3. Under Linker Input, add "CLIWrapperDLLd.lib", for "Additional Dependencies", add "CLIWrapperDLLd.lib" for debug configuration and "CLIWrapperDLL.lib" for release configuration.
4. For MFC projects, under Linker – Input, for the "Delay Loaded Dlls", add "CLIWrapperDLL.dll" (CLIWrapperDLLd.dll for debug configuration), otherwise you might get a runtime exception.

In addition, there are DLLs needed at runtime. Typically these will be installed in the GAC (by a complete 835 Suite installation). Since Visual Studio does not allow you to look directly in the GAC for the references, you will need to manually copy these dlls from <install directory>[3]\API and the CLIWrapperDLL.dll files\Net to the directory with the executable. Windows will use the GAC files automatically when the application executes so this is only necessary for developing and debugging your application.

## 6.1 Using CLIWrapperDLL

There are 2 export functions defined within the CLIWrapperDLL namespace :-

1. *CreateServiceHandle*
   This should be the very first call into the DLL, to create and return an IServiceWrapper interface that provides all the functionality to communicate with a Controller. Any exception thrown with this call is probably due to a missing DLL at runtime. See previous section on the dependent runtime DLLs.
2. *ReleaseServiceHandle*
   This call properly disposes of the IServiceWrapper interface that was created with CreateServiceHandle. It should be called to prevent memory leaks. This typically should be the last call before the client application exits, or whenever the IServiceWrapper interface is no longer needed.

[3] <install directory>: C:\Program Files (x86)\GP\VQM or C:\Program Files\GP\VQM

[4] <public documents> is the shared document location for the specific Windows operating system

C:\Users\Public\Documents\VQM\ErrorLogs (Windows 7) or C:\Documents and Settings\All Users\Documents\VQM\ErrorLogs (Windows XP)

## 6.2 IServiceWrapper

This interface is defined in IServiceWrapper.h, and it consists of functions that are essentially redirect calls into the CSServiceWrapper DLL. Please refer to the previous sections of this dcoument for the functions' detailed description.

## 6.3 IHeaderData

This interface is defined in IHeaderData.h and it provides all the getter functions to properties of the header data. The IServiceWrapper has 2 functions that return this interface and the caller must call IHeaderData->Release() to dispose the object when it is no longer needed to prevent memory leaks.

This interface also returns the following interfaces: -

- IADCOutput – provide getter functions to properties of the ADC output.
- IMassAxis – provide getter functions to properties of the Mass Axis data.
- INoiseData – provide getter functions to properties of the Noise Data.

IHeaderData->Release() implicitly invokes the Release() calls of each these 3 interfaces.
There are currently no functions in IServiceWrapper that return any of these 3 interfaces individually.

## 6.4 Structures

The following structures are simple containers for the data retrieved from the CSServiceWrapper DLL, and the implementation is in the .h files.

- SAnalyzedData
- SAverageData
- SComboFileData
- SComboFileGasData
- SVQM_800_Error
- SRawConnectionOptions

## 6.5 Code Example

```
#include “IServiceWrapper.h”
#include “SVQM_800_Error.h”

try
{
	IServiceWrapper* serviceWrapper  = CLIWrapperDLL::CreateServiceHandle();
	if (serviceWrapper != NULL)
	{
		SDeviceConnectionInfo* buffer = NULL;
		int nConnections;
		SVQM_800_Error error = serviceWrapper->GetValidConnections(nConnections, buffer);
		if ( error.m_ErrorType != NO_ERRORS)
		{
			//There are no valid connections. Do reporting here.
		}
		else
		{
			if (buffer != NULL)
			{
				for (int i = 0; i < nConnections; ++i)
				{
					//Display list of connections
				}
				delete[] buffer;
			}
		}
		CLIWrapperDLL::ReleaseServiceHandle(serviceWrapper );
	}
}
catch ( wchar_t* str )
{
	//Exception instantiating IServiceWrapper, most probably due to missing runtime DLLs
}
```

# 7 Visual Basic Examples

The following are Visual Basic examples for the .NET application programming interface. These examples are snippets of code from test programs to demonstrate the Visual Basic use of the major API members documented in previous sections.

The examples in this section are grouped into the following categories:

- **Connections (7.1 )** – an example of connecting to the service and a Controller and an example of disconnecting.
- **Errors (7.2 )** – example shows how to parse the error returned from the API calls including getting more details (if available) from the Controller itself. The examples in this section call the LookforErrors subroutine for displaying the error and troubleshooting messages.
- **Scanning (7.3 )** – example of starting the actual mass spec scan after checking to be sure that the total pressure is in the safe range and an example of stopping the Controller scanning.
- **Analysis (7.4 )** – examples show how to set the averaging mode and how get fitted gas results.
- **Logging (7.5 )** – examples of starting and stopping logging of mass spec data.
- **Getting or setting configurable Controller settings (7.6 )** – Examples of getting the current voltage, the minimum and maximum voltage and the setpoint voltages for the mass spec logical instruments, the electron multiplier electrometer gain, and the filament current. The examples also show setting the setpoint for the voltages, electrometer gain, and filament current.
- **Configuration setting and NVRAM (7.7 )** – examples of restoring user default, restoring factory defaults as the current settings and and example of saving current configurable settings as user settings.

## 7.1 Connections

To use the API to the Controller:

Create a new CServiceWrapperManager for connection to the service.

Create a VQM_800_ERROR variable for the api function return.

Create a new CDeviceConnectionInfo for the information about the Controller connection.

```
Private api As New CServiceWrapperManager()
Protected err As Utilities.VQM_800_ERROR
Protected controllerList() As CDeviceConnectionInfo
Protected connInfo As New CDeviceConnectionInfo()
```

The first step is to connect to the Controller service to get the list of available controllers. Controller must be connected to PC USB port and powered on. GetValidConnections will list the COM ports with Controller availability. It has optional parameters to specify another host reachable by a TCP connection.

```
err = api.GetValidConnections(controllerList) err is a Utilities.VQM_800_ERROR
If (Not err.ErrorType = VQM_800_ERROR.no_error) Then
        LookforErrors(err)
Else
      ' Add the list of Controller returned to the Controller list box
        cntlListBox1.Items.Clear()
        For Each d As CDeviceConnectionInfo In controllerList
            Dim s As String
            s = d.Availability.ToString() + " " +
            d.DeviceAddress.ToString() + " " + d.DeviceType.ToString
            () + " " + d.SerialNumber.ToString()
            cntlListBox1.Items.Add(s)
        Next
End If
```

Getting the list from a remote host, TEST02. An IP address can be used instead of the name in the string.

```
err = api.GetValidConnections(controllerList,
          CDeviceConnectioninfo.EServiceConnectionType.TCP,”TEST02”) err is a
          Utilities.VQM_800_ERROR
```

Connect to the selected Controller. To Call the ConnectToDevice API you need the specific CDeviceConnectionInfo from the list that you wish to connect. Once connected this connInfo is a parameter to all of the other API functions.

```
    Dim isMaster As Boolean
    connInfo = controllerList(cntlListBox1.SelectedIndex)
    connInfo.DeviceType =
    ServiceWrapper.CDeviceConnectionInfo.EConnectionType.VQM_835
    err = api.ConnectToDevice(connInfo, isMaster)
   ' set global varible to connected state
    If err.ErrorType = VQM_800_ERROR.no_error Then
        connected = True
        MessageBox1.Text += vbCrLf + "Connected"
        If isMaster Then
             MessageBox1.Text += " as master"
        Else
             MessageBox1.Text += " as a monitor"
        End If
    Else
        connected = False
        LookforErrors(err)
    End If
```

Example of disconnecting from the Controller as when exiting the application. Note, disconnecting from the Controller does not change the state (e.g. settings, scanning, etc.) of the Controller. If exiting from the application the CloseConenction should also be called.

```
    err = api.DisconnectFromDevice(connInfo)
    If (Not err.ErrorType = VQM_800_ERROR.EErrorType.NO_ERROR) Then
            MessageBox1.Text += vbCrLf + "Error disconnecting"
            MessageBox1.Text += vbCrLf + err.ErrorString
    End If
    ‘when exiting the application also call the api.CloseConnection
    err = api.CloseConnection()
```

## 7.2 Errors

Example of getting the error message and troubleshooting tip from the VQM_800_ERROR returned by the service API call. This example also provides the Controller error queue messages.

```
Private Sub LookforErrors(ByVal rtnErr As VQM_800_ERROR)
Dim errCnt As Integer
Dim errQueue(20) As String
If (Not rtnErr.ErrorType = VQM_800_ERROR.no_error) Then
     MessageBox1.Text += vbCrLf + "Error returned: " + rtnErr.ErrorString
     MessageBox1.Text += vbCrLf + "Try: " + rtnErr.ErrorTroubleShooting
     err = api.GetErrorCountAndQueue(errCnt, errQueue, connInfo)
     If (errCnt > 0) Then
        MessageBox1.Text += vbCrLf + "Further details from Controller error
        queue:"
        For Each s In errQueue
            MessageBox1.Text += vbCrLf + "   " + errQueue(s)
        Next s
      End If
End If
End Sub
```

## 7.3 Scanning

Example of turning the gauge on. There are two ways to do this – either TurnOnGauge or SetGaugeState, but SetGaugeState is preferred.

```
err = api.SetGaugeState(EGaugeState.ON, connInfo)
If (Not err.ErrorType = VQM_800_ERROR.no_error) Then
    LookforErrors(err)
End If

Or
err = api.TurnOnGauge(connInfo)
```

Expanded example of checking for safe pressure range before turning on the gauge and scanning.

```
    Dim totPress As Double
    Dim units As String
```

Example of setting the source for the total pressure measurement to the 390.

The Controller must be powered on to get the external total pressure from the 390802 (RS-485).

(The string must match a string in the list returned from *api.Data AnalysisGetPressureSourceStrings*.

```
err = api.DataAnalysisSetPressureSource(Pressure_Source._390_TPMK, connInfo)
If Not err.ErrorType = VQM_800_ERROR.no_error Then
    LookforErrors(err)
End If
 ' It is also a good idea to be sure that the averaging mode is set so you
 ' get the averaged external pressure
err = api.DataAnalysisSetAvgMode(glAvgMode, connInfo)
If Not err.ErrorType = VQM_800_ERROR.no_error Then
     LookforErrors(err)
End If
err = api.DataAnalysisSetNumAvgs(numAverages, connInfo)
If Not err.ErrorType = VQM_800_ERROR.no_error Then
     LookforErrors(err)
End If
```

Example of getting the External Total Pressure. There are two different ways to get the External Total pressure: 1) using the GetExternalTotalPressure, and 2) using the Header data from the fetches. The GetExternalTotalPressure returns the pressure in Torr for the current pressure source (either 390802 or a supported analog pressure gauge when the pressure source has been set.

```
err = api.GetExternalTotalPressure(totPress, connInfo)
or
Dim stateData As Utilities.HeaderData835
err = api.GetHeaderData(stateData, connInfo)
```

Also, need to get the units for the External Pressure to make sense of the double returned from the GetHeaderData API call. This is an example of getting the pressure and pressure units for the 390:

```
If err.ErrorType = VQM_800_ERROR.no_error Then
    totPress = stateData.TotalPressure
    MessageBox1.Text += "No error getting 390 total pressure"
    units = stateData.ETPressureUnits
   ' If the units are not in torr, convert to torr
   ‘ If units are None then total pressure is invalid
    If (unit.Contains(“NONE”
    If (units.Contains("pa")) Then
         totPress = totPress * 0.0075
```

```
    ElseIf (units.Contains("mbar")) Then
         totPress = totPress / 1.3332
    End If
Else
    LookforErrors(err)
End If
```

For non-supported gauges, get the Analog In Voltage and convert to pressure units. See the VQM System manual (p/n 835000) for instructions to add pressure sources to the list.

```
err = api.GetAnalogInputVoltage(out analogInVolts, connInfo)
If err.ErrorType = VQM_800_ERROR.no_error Then
    totPress = conversion of voltage to pressure
End If
```

Now that pressure is known to be in correct range, set the scan range, turn the gauge on, and start the Controller continuously scanning. You must be connected as a master to do this.

```
Dim lowerRange As Double = 1.0
Dim upperRange As Double = 120.0
' Lowest total pressure should be 1E-15 (1E-99 is not initialized - probably
' no external pressure gauge)
If ((totPress > 0.000000000000001) And (totPress < 0.00001)) Then
    err = api.SetScanRange(lowerRange:=lowerRange,upperRange:=upperRange,
    connectInfo:=connInfo)
    if err.ErrorType = VQM_800_ERROR.no_error then
        err = api.SetGaugeState(EGaugeState.SCAN, connInfo)
        If err.ErrorType = VQM_800_ERROR.no_error Then
            scanning = True
        Else
            scanning = False
            LookforErrors(err)
        End If
    Else
        MsgBox(“Error setting scan range”)
Else
     MsgBox("Total Pressure is not within acceptable range for using the MSI
gauge")
End If
```

Fetch the mass spec data.

```
Dim results As New Utilities.AnalyzedData()
Dim avgData As New Utilities.AverageData()
Dim stateData As New Utilities.HeaderData835
Dim isValidData As Boolean
Dim gaugeState as EGaugeState
Dim scanNum As Integer = 0   ‘ initial value after that can use value
                             ‘ from previous scan
err = api.GetScanData(scanNumber:=scanNum, results:=results,
currentStateData:=stateData, averageData:=avgdata, connectInfo:=connInfo,
isValidData:=isValidData, controllerState :=gaugeState)
If Not err.ErrorType = VQM_800_ERROR.no_error Then
    LookforErrors(err)
End If
```

If you are interested in only the processed data, *use err=api.GetProcessedScanData* (scan number:=ScanNum, results:=results, average pressure data:=AveragePressureData, connect info:=ConnInfo).

Example of using the API to stop the Controller from scanning. To use command you must be the "master" to control the Controller and not a "monitor" who only gets to read (fetch) data. The Master connection can use the SetGaugeState to turn the gauge off in a single call and SetGaugeState is the preferred method as StopScan has been deprecated.

```
err = api.StopScan(connInfo)
 or
err = api.SetGaugeState(EGaugeState.ON)
If (Not err.ErrorType = VQM_800_ERROR.no_error) Then
    LookforErrors(err)  ' Will see "insufficient privilege" error message if ' you
are a "monitor"
End If
```

Example of using the API to turn off the gauge. To use this command you must be connected as the "master". SetGaugeState is the preferred method.

```
or
err = api.SetGaugeState(EGaugeState.OFF)
If (Not err.ErrorType = VQM_800_ERROR.EErrorType.NO_ERROR) Then
    LookforErrors(err)
End If
Or
err = api.TurnOffGauge(connInfo)
```

## 7.4 Analysis

Example code for setting the averaging mode for the Data Analysis. The averaging mode is an enumeration type which includes: no averaging, running average, cumulative moving average, and accumulator averaging.

Using N as the number of scans to average, Running average (AKA Finite Impulse Response (FIR) filter): the N most recent scans are summed and divided by N  N can be from 2 to 100 scans.

Cumulative Moving Average (AKA Infinite Impulse Response (IIR) filter): (N-1)*previous average + current scan/N. N can be from 2 to 50,000.

Accumulator: Sum N scans and divide by N A new value is reported every N scans.

Unlike the running average, there are no intermediate values. N can be from 2 to 50,000 scans.

```
Dim avgMode As Utilities.Avg_Mode
 Dim avgIndex As Integer = avgTypeBox1.SelectedIndex
 Select Case avgIndex
     Case 0
                avgMode = Utilities.Avg_Mode.Off
                numAverages = 1
     Case 1
                avgMode = Utilities.Avg_Mode.Accumulator
     Case 2
                avgMode = Utilities.Avg_Mode.Cumulative_Moving_Avg
     Case 3
                avgMode = Utilities.Avg_Mode.Running_Avg
     Case Else
                avgMode = Utilities.Avg_Mode.Off
 End Select
 ' The averaging history is cleared when you set averaging mode to a new mode
   err = api.DataAnalysisSetAvgMode(avgMode, connInfo)
If (Not err.ErrorType = VQM_800_ERROR.no_error) Then
    LookforErrors(err)
End If
```

Example code to set the number of scans to average when averaging is turned on (i.e., any mode except Off).

```
Dim numAverages As Integer = noAvgBox1.Text
err = api.DataAnalysisClearAvgHistory(connInfo)
err = api.DataAnalysisSetNumAvgs(numAverages, connInfo)
If (Not err.ErrorType = VQM_800_ERROR.no_error) Then
    LookforErrors(err)
End If
```

Example of using the fitted gases part of results data returned from GetScanData. First use the GetComboFileData to get the list of gases.

```
Dim gasFileData as ComboFileData
err = api.getComboFileData(gasFileData, connInfo)
If (Not err.ErrorType = VQM_800_ERROR.no_error) then
    LookForErrors(err)
Else
    For Each gas in gasFileData.List
        msgBox("Gas is " + gas.GasName.ToString())
    Next
End If
err = api.GetScanData(scanNumber:=scanNum, results:=results,
currentStateData:=stateData, averageData:=avgdata, connectInfo:=connInfo)
If (Not err.ErrorType = VQM_800_ERROR.no_error) Then
    LookforErrors(err)
Else
    For Each gas In results.FittedGases
        fittedGases(gas) = results.FittedGases(gas)
    Next gas
    currAvgPress = avgdata.AvgTotalPres
```

In the current gas library (ComboDataFile), item 5 is hydrogen and item 9 is water. Use the ratio of these to indicate whether the atmosphere in the vacuum chamber is a reducing atmosphere.

```
    If fittedGases(9) > 0.0 Then
        currentRatio = fittedGases(5) / fittedGases(9)
        If (currentRatio >= Ratiogoal And currAvgPress < TPmax) Then
            NotifyTargetAchieved(currAvgPress, currentRatio)
        End If
    End If
End If
```

## 7.5 Logging

Example of using the API to start raw logging. The API call parameters are the connection information and an optional string folder path for the log file(s). If the optional string folder path is used, the folder must already exist and the user must have permissions to write to that folder. The installer creates a default folder location for the default logFolderPath.

```
err = api.StartLogging(connInfo)  log to default location
or
err = api.StartLogging(connInfo, logFolderPath:="C:\temp") log to c:\temp folder
If (Not err.ErrorType = VQM_800_ERROR.no_error) Then
    LookforErrors(err)
End If
```

Example of using the API to stop logging.

```
err = api.StopLogging(connInfo)
If (Not err.ErrorType = VQM_800_ERROR.no_error) Then
                LookforErrors(err)
End If
```

## 7.6 Get/Set Configuration Data

Example of using GetHeaderData. Create a CCurrentStateData variable to be able to get the current values returned in the Fetch header from the Controller. This state data includes current values for voltages, electometer gain, and filament emission current, and the external total pressure (if a TPMK is connected to the Controller.) However, if you are not scanning the current values will not be close to the setpoints. Get the Header data with the current values in it. Then parse out the appropriate values and convert to strings for display.

```
Dim currStateData As Utilities. HeaderData835
Dim currFilEms, setptFilEms, minFilEms, maxFilEms As Double
Dim minMax As CLogicalInstrumentMinMax
Dim ExtTotalPressure As Double
err = api.GetHeaderData(currStateData, connInfo)
If (Not err.ErrorType = VQM_800_ERROR.EErrorType.NO_ERROR) Then
            LookforErrors(err)
Else
    currValue2.Text = currStateData.FilamentBiasVoltage.ToString()
    currValue3.Text = currStateData.RepellerVoltage.ToString()
    currValue4.Text = currStateData.EntryPlateVoltage.ToString()
    currValue5.Text = currStateData.PressurePlateVoltage.ToString()
    currValue6.Text = currStateData.CupsVoltage.ToString()
    currValue7.Text = currStateData.TransitionVoltage.ToString()
    currValue8.Text = currStateData.ExitPlateVoltage.ToString()
    currValue9.Text =
    currStateData.ElectronMultiplierShieldVoltage.ToString()
    currValue10.Text = currStateData.ElectronMultiplierVoltage.ToString()
```

Example of getting the instantaneous total pressure from the external TPMK, the pressure units for the external TPMK - these are in the CurrentStateData structure returned from the api GetHeaderData call.

```
   TPBox1.Text = currStateData.TotalPressure.ToString
   TPUnitsBox1.Text = currStateData.ETPressureUnits.ToString
End If
```

Example of getting the total pressure using the GetExternalTotalPressure. (pressure displayed in Torr)

```
err = api.GetExternalTotalPressure(ExtTotalPressure, connInfo)
If (Not err.ErrorType = VQM_800_ERROR.EErrorType.NO_ERROR) Then
    LookforErrors(err)
End If
```

Get the setpoint, then the minimum, and then the maximum values for the Filament Bias Voltage. Voltage for the other logical instruments are gotten the same way using the appropriate ELogicalInstruments enum.

```
Dim setpointFilBV As Double
Dim minMax As CLogicalInstrumentMinMax
err = api.GetLogicalInstrumentVoltageSetpoint(Utilities.ELogicalInstruments.
      FilamentBias, setpointFilBV, connInfo)
err = api.GetLogicalInstrumentMinMaxVoltage(Utilities.ELogicalInstruments.
      FilamentBias, minMax, connInfo)
LookforErrors(err)
setpoint.Text = setpointFilBV.ToString()
minValue.Text = minMax.MinValue.ToString()
maxValue.Text = minMax.MaxValue.ToString()
```

Get the current value, setpoint, and the minimum and maximum values for the Filament Emission current. Filament Emission Current is not returned in the the GetHeader or the GetLogicalInstrument voltage calls.

```
Dim currFilems, setptFilEms As Double
```

```
Dim minMax As CLogicalInstrumentMinMax
err = api.GetFilamentEmissionCurrent(currFilEms, connInfo)
err = api.GetFilamentEmissionCurrentSetpoint(setptFilEms, connInfo)
err = api.GetFilamentEmissionCurrentMinMax(minFilEms, maxFilEms, connInfo)
currValue.Text = currFilEms.ToString()
setpointValue.Text = setptFilEms.ToString()
minValue.Text = minFilEms.ToString()
maxValue.Text = maxFilEms.ToString()
```

Get the setpoint, then the minimum, and maximum values for the EM Bias Voltage. The EM Bias voltage acts as a signal gain.

```
err =
api.GetLogicalInstrumentVoltageSetpoint(Utilities.ELogicalInstruments.EMBias,
setVB, connInfo)
err = api.GetLogicalInstrumentMinMaxVoltage(Utilities.ELogicalInstruments.EMBias,
minMax, connInfo)
setpointValue.Text = setVB.ToString()
minValue.Text = minMax.MinValue.ToString()
maxValue.Text = minMax.MaxValue.ToString()
```

Get the current value, setpoint, then the minimum, and then the maximum values for the RF Amp (Peak to Peak) logical instrument.

```
err = api.GetLogicalInstrumentCurrentVoltage(Utilities.ELogicalInstruments.RFAmp,
currVB, connInfo)
err = api.GetLogicalInstrumentVoltageSetpoint(Utilities.ELogicalInstruments.RFAmp,
setVB, connInfo)
err = api.GetLogicalInstrumentMinMaxVoltage(Utilities.ELogicalInstruments.RFAmp,
minMax, connInfo)
currValue.Text = currVB.ToString()
setpointValue.Text = setVB.ToString()
minValue.Text = minMax.MinValue.ToString()
maxValue.Text = minMax.MaxValue.ToString()
```

Example of getting the current value of the Mass Calibration Factor.

```
Dim currCal As Double
Dim minMax As CLogicalInstrumentMinMax

err = api.GetMassCalibrationFactor(currCal, connInfo)
currValue.Text = currCal.ToString()
err = api.GetMinMaxMassCalibrationFactor(minMax, connInfo)
minValue.Text = minMax.MinValue.ToString()
maxValue.Text = minMax.MaxValue.ToString()
```

Example of getting Electron Multiplier Gain values. Use the current state data from the header for the current value of the Electron multiplier Electrometer Gain. Use the specific calls to get the setpoint, minimum and maximum for the Electron Mulitplier Electrometer Gain.

```
Dim setPGain, minGain, maxGain As Double
err = api.GetHeaderData(currStateData, connInfo)
If (Not err.ErrorType = VQM_800_ERROR.EErrorType.NO_ERROR) Then
    LookforErrors(err)
Else
    currValue.Text = currStateData.ElectronMultiplierElectrometerGain.ToString()
End If
err = api.GetElectrometerGainSetpoint(setPGain, connInfo)
err = api.GetElectrometerGainMinMax(minGain, maxGain, connInfo)
setpointValue.Text = setPGain.ToString()
minValue.Text = minGain.ToString()
maxValue.Text = maxGain.ToString()
```

Example of changing the setpoint for the filament emission current.

```
Dim value As Double
value = Convert.ToDouble(setpointValue.Text)
err = api.SetFilamentEmissionCurrentSetpoint(value, connInfo)
If (Not err.ErrorType = VQM_800_ERROR.EErrorType.NO_ERROR) Then
    LookforErrors(err)
End If
```

Examples of changing the setpoint for the filament bias voltage. One of the logical instrument voltages.

```
Dim value As Double
value = Convert.ToDouble(setpointValue.Text)
err = api.SetLogicalInstrumentVoltageSetpoint(Utilities.ELogicalInstruments.
      FilamentBias, value, connInfo)
If (Not err.ErrorType = VQM_800_ERROR.EErrorType.NO_ERROR) Then
    LookforErrors(err)
End If
```

Change the Mass Calibration factor setting - tuning the mass/charge spectrum.

```
Dim value As Double
value = Convert.ToDouble(setpointValue.Text)
err = api.SetMassCalibrationFactor(value, connInfo)
If (Not err.ErrorType = VQM_800_ERROR.EErrorType.NO_ERROR) Then
               LookforErrors(err)
End If
```

Example of changing the setpoint for the Electron Multiplier Electrometer Gain.

```
Dim value As Double
value = Convert.ToDouble(setpointValue.Text)
err = api.SetElectrometerGainSetpoint(value, connInfo)
If (Not err.ErrorType = VQM_800_ERROR.EErrorType.NO_ERROR) Then
    LookforErrors(err)
End If
```

## 7.7 Configuration Setting and NVRAM

Example of restoring the user defaults as the current setpoint values for the connected Controller.

```
err = api.RestoreUserSettings(connInfo)
 If (Not err.ErrorType = VQM_800_ERROR.no_error) Then
     LookforErrors(err)
 End If
```

Example of restoring the factory default settings as the current setpoint values for the connected Controller.

```
err = api.RestoreFactoryDefaults(connInfo)
If (Not err.ErrorType = VQM_800_ERROR.no_error) Then
    LookforErrors(err)
End If
```

Example of using the API to save the current Controller settings to NVRAM.

```
err = api.SaveUserSettingsToEEPROM(connInfo)
If (Not err.ErrorType = VQM_800_ERROR.no_error) Then
    LookforErrors(err)
End If
```

## 7.8 Auto Tune

Example of using the API to start Auto Tune of the connected gauge. Must be the master connection to do this. The defaults are Complete Auto Tune and Normal Gain Target.

```
Dim logFName As String = “c:\\data\VbAutotune.log”
err = api.StartAutotune(logFName, connInfo)
if (Not err.ErrorType = VQM_800_ERROR.noError) then
   ‘error message
End If
```

Alternative examples are:.

```
err = api.StartAutotune(logFName, connInfo)mode as utilities.EAutotuneType, target
peak height as Utilities.EAutotuneEMBiasTargetPeakHeight)

err = api.StartExpressAutotune(logFName, connInfo)mode as utilities.EAutotuneType,
target peak height as Utilities.EAutotuneEMBiasTargetPeakHeight)

err = api.StartJustEMAutotune(logFName, connInfo)mode as utilities.EAutotuneType,
target peak height as Utilities.EAutotuneEMBiasTargetPeakHeight)
End If
```

Example of using the API to cancel a running Auto Tune process. Must be the master connection to do this.

```
err = api.CancelAutotune(connInfo)
If (Not err.ErrorType = VQM_800_ERROR.noError) then
   ‘error message
End If
```

Example of using the API to get the progress or state of a running Auto Tune process. May be a monitor connection to do this.

```
Dim aStatus As EAutotuneStatus
Dim percentDone As Integer
Dim EMBias, RepBias, ExitBias, rfAmp as Double
err = api.GetAutoTuneStatus(status:=aStatus, percentDone:=percentDone,
emBias:=EMBias, repeller:=RepBias, exitPlate:=ExitBias, shield:=EMSHBias,
rfamp:=rfAmp, connectInfo:=connInfo)
If (Not err.ErrorType = VQM_800_ERROR.noError) then
   ‘error message
Else
    statusTextBox.Text = aStatus.ToString()
    percentTextBox.Text = percentDone.ToString()
End If
```

# 8 Troubleshooting

## 8.1 USB 3.0 Ports

Some PCs, particularly laptops, provide USB 3.0 ports, but the 835 VQM supports USB 2.0. While in general USB 3.0 is backwards compatible with USB 2.0 some PC implementations are not fully compatible. This causes the communication between the controller and PC to be fragile or even fail to connect at times. If the PC in question is one of these and does not provide USB 2.0 ports to use you may be able to disable the USB 3.0 controller in the BIOS. See the troubleshooting section of the 835 VQM Instruction Manual for details.

## 8.2 Get Valid Connections List is Empty

The list of connected controllers returned from GetValidConnections will only contain controllers that are powered on so verify that the controller is not in Standby power mode and that the USB led on the controller is illuminated. Use the Windows device manger to verify that the operating system has recognized the controller and loaded the correct device driver. The device manger should show a COM port with Granville-Phillips VQM 835 under "Ports (COM & LPT)" (as in Figure 4) if the device driver for the controller has been loaded properly.

If the controller is not recognized, the device manager may show the following:

- The VQM 835 is missing
- The VQM 835 is listed but indicates a yellow triangle with an exclamation point (not working properly)
- There is an "unknown device"

If either of these situations occur, power cycle the controller to try again. You may need to reinstall the 835 driver – see installation of the USB driver in the 835 VQM Instruction Manual.

If the device manager shows the com port with the Granville-Phillips VQM, use a serial communication program such as Hyperterminal or Termite to attempt to connect to the controller and then disconnect to free the port. If the serial communication program can connect, but GetValidConnections still does not list the controller then you may have a corrupt WMI on your PC or are connected to a USB 3.0 port that is not compatible with the controller (see Section 8.1 ).

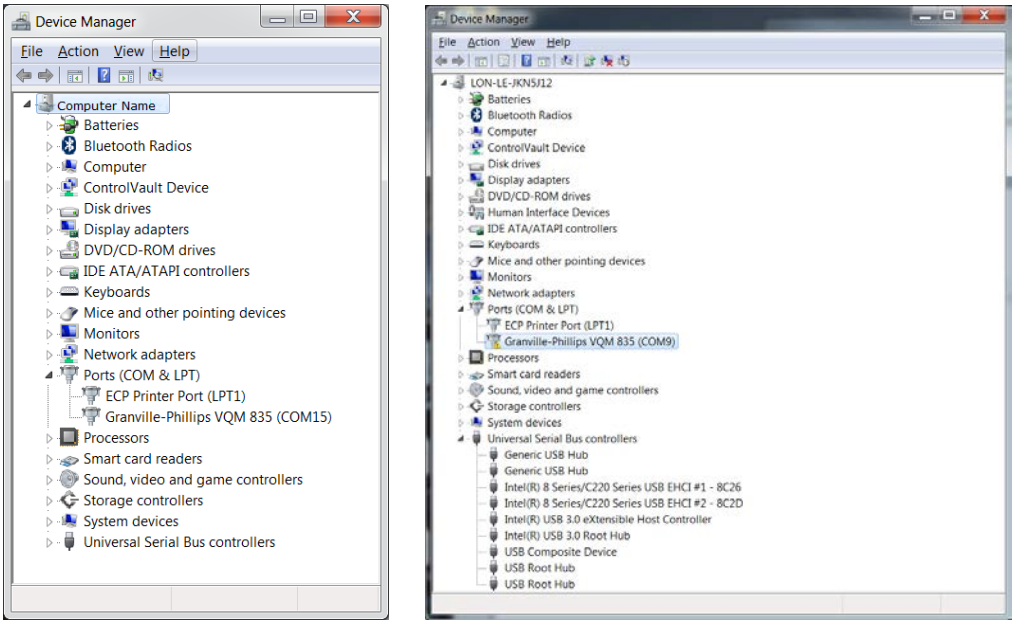



**Figure 4: Device Manager recognizes VQM 835 & Device Manager indicates Not Working Properly**

## 8.3 Unable to Connect

Local connections to a Service can best be debugged by testing the desired behavior with the Viewer Application. That is, can the Viewer Application connect, disconnect, set controller settings, and scan? If so, then you know that the Service talking to the controller is operational.

If you can not get the Viewer Application to see the controller or to see it as available, you can try restarting the Services. You can also try power cycling the controller or unplugging and replugging the USB cable (verifying that the correct driver is found) with power cycling the controller.

## 8.4 No Gauge Settings Control

If you can connect successfully, but can not control the gauge (changing from off to on, standby, scanning, etc.), perhaps you do not have a master connection. When you connect, the parameter isMaster is true if you have master permissions. It is false if you do not. If you are the first application to connect to the controller, you should have master permissions. However, if no master is connected, but an application has a monitor connection, no application will be able to become the master until all of the connections are terminated.

Occasionally, you may find that a program exited in such a way that the Service has maintained a connection, even though the application that established it is gone. In this case, you may need to restart the Service. You can use the Utility "800SeriesServiceRestart.cmd" in the Utilities directory of the install location. This will restart all the services. If you would rather restart just one, you can find the commands to do that in the script file or you can use Windows administrative tools to restart a particular Service. Both of these methods for restarting Services require administrator privileges.

## 8.5 Lost Connection

The use of a debugger can complicate timeouts that are built into the Services. For the most part you will have no problems. However, a Remote connection that is halted in the debugger may find that its connection is terminated by a timeout. Try to avoid using breakpoints for prolonged time periods. Consider logging or displaying debugging information. You may need to reconnect if you lose the connection due to timeout. Excessive use of debug logging statements can cause enough delay in some situations to trigger a timeout. Use them sparingly in released versions of your application.

Connections can be lost for USB, Controller, or Windows problems. Verify that the Viewer Application is seeing the same kinds of unexpected disconnects. If it is, you'll need to troubleshoot the specific failure. The Service maintains a error log for each running Service in the <public documents>[4] directory. Determine which of these is associated with your troublesome connection and check the error messages. These may help guide you in your efforts.

Occasionally, you may find that a program exited in such a way that the Service has maintained a connection, even though the application that established it is gone. In this case, you may need to restart the Service. You can use the Utility in <install directory>[3]Utilities\800SeriesServiceRestart.cmd. This will restart all the services. If you would rather restart just one, you can find the commands to do that in the script file or you can use Windows administrative tools to restart a particular Service. Both of these methods for restarting Services require administrator privileges.

If the target computer can not keep up with the requests from your application, the messages will by queued by Windows. We have tried to configure the Services to handle the load, but the actual ability to keep up is based on your computer. If the Service is not keeping up, you would notice that you can successfully connect and query status, but when you start methods which enable close periodic calls (like GetScanData), you experience a disconnect after a few minutes. Note this is not necessarily the connection timeout. Try adding Sleeps between the periodic calls that are excessive (2 seconds, for example) and see if the situation recovers. If it does not, then it is unlikely that you are exceeding the Service's ability to keep up with requests.

## 8.6 Controller Errors

Errors generated by the Controller (type 5014) are detailed via calls to GetErrorCountAndQueue. The error string returned will indicate the problem, e.g., a parameter out of range, the gauge was in the wrong state for the command, etc.

The TPMK (390802), if used, may generate errors also. These error conditions are returned in the Header information. The user is responsible for monitoring the 390802 status and for performing associated behaviors. The Service monitors the 390 status as well to assure that the Total Pressure is valid.

## 8.7 Developing your own Data Analysis functions

The most common functions you will need for Data Analysis are denoising algorithms and peak finding/fitting algorithms. The API GetScanData provides access to the raw ADC readings, averaged ADC readings, denoised data, and fitted peaks. The Histogram height is based on the area under the fitted peak. Those values are stored in results.PeakArea. GetScanData also provides access to gas-fitting results along with residuals (leftovers after gas-fitting).

Your own data analysis functions can use any of these outputs as inputs. However, the API does not allow you to apply your outputs as inputs to the 835's data analysis functions. For example, if you use the 835's averaged data as an input to your own denoising algorithm, you will probably want to provide your own peak fitting and finding also.

The one thing to remember about peak amplitude is that the prevalence of a mass peak is not proportional to its fitted peak height, but to the area under the fitted peak. You can see this on the Tune Screen of the Viewer Application on the Normalized graph. Zoom in on one of the peaks. If you are looking at the main peak, you may find that the curve and the histogram are the same height. However, most of the other peaks will show that their curve heights and histogram heights are different. The histogram height represents the area under the curve (normalized). The curves for peaks at higher amus are wider, and therefore have more area under the curve for the same height curve.

## 8.8 Crashes on Exit

The sequence for exiting your application should include calls to disconnect from any connected devices and to close the service connection. Various threads have to complete their activity triggered by DisconnectFromDevice before the CloseConnection call is made. Therefore, we recommend you call DisconnectFromDevice, Sleep 1000 milliseconds (1 second), and then call CloseConnection. (See Section 2.1 and Figure 2.) This gives the various threads that need to respond to the Disconnect time to complete their activities before you remove their associated object (via CloseConnection.)

If your requests to the Service take longer to resolve, you may need to either keep track of open requests or extend the Sleep. Consult the Windows Event Viewer Application Logs for details regarding crashes.

## 8.9 Performance Issues

### 8.9.1 Your Computer

A couple of things can prevent fast performance. The biggest one is the computer specifications. The software package will allow connections to multiple controllers and multiple connections to the same computer. However, only you can determine what kind of computer you can tolerate based on your use.

### 8.9.2 Multiple Connections/GetScanData

If you have multiple connections calling GetScanData from a single controller, you will find that the calls stack up behind each other. That is, there is a single published results set that each of the connections needs access to. The reads of the published data set all occur in the same thread. This means that a monitor that is calling GetScanData will block a master that is calling GetScanData until the monitor's new scan data is available. To assure that the block lasts as short a time as possible, all callers of GetScanData and other calls that use scanNumber as a parameter should assure that the scanNumber input is equal to the last returned scanNumber. That is, do not increment it. Use the return value from the previous call. Initially, use a scanNumber of 0. GetScanData will return the first scan that does not have the same scanNumber as the one you input. Using any number other than the return value may cause unnecessary delays.

Additionally, certain controller values are available in the data fetched every scan. Other values are not. If your application reads values that are not available in the scan data, separate requests for the data are sent to the controller. These requests take time and will block requests for scan data until after they are completed. You may want to limit requests for non-scan data to optimize the amount of time available for transfering the scan data to the PC. For example, voltage setpoints are not returned with scan data. A request for a voltage setpoint will block the request for scan data until the setpoint's request reply is received.

### 8.9.3 Windows Indexing Service

Both Windows XP and Windows 7 provide an indexing service that scans your disk to find available files and to determine their associations. This service will use CPU cycles, and, depending on how many files are on your drive, the time used can be excessive. You may want to disable the Indexing Service. On XP, use the Administrative Tools to disable the service. You will need to restart after disabling the service. On Windows 7, in the Control Panel, navigate to Programs and Features and disable Windows Indexing Service feature.

### 8.9.4 Scan Times

Because of the blocking nature of GetScanData and similar methods, you probably want to restrict how often you call them to a number that supports the application. For example, a health monitor may need to call GetScanData once an hour. A plotting program will need to call often enough to maintain the graphing function. Neither should try to call GetScanData significantly faster than the scan rate of the controller. The scan rate of the controller is a function of the mass range that is scanned. One way to tell the approximate scan rate is to run the Viewer Application. Go to the Tune Screen and view the Diagnostic information. Near the bottom of the pane you will see HW Loop Time. This is the approximate scan time. You may want to put a "Sleep" call between calls to GetScanData that is based on scan time.

### 8.9.5 Your EtherNet

If you are running a remote session, your EtherNet performance will impact the application's performance also. If this is an issue for you, you can consider running on a private network.

### 8.9.6 Missed Scans, Discarded Fetch Data, Scan Timeouts

When you look at the Service Log to help determine performance issues, you may find statements that indicate that Fetch Data has been discarded or GetAnalyzedData timed out.

GetAnalyzedData time-outs are usually associated with multiple connections to the same controller (a master and a monitor, for example). If one of the connections is slow, the other connection may see Scan Timeouts. These are not fatal, although they may be annoying or important, depending on your application. If you want to minimize the number of Scan Timeouts, consider changing how often you request data. The slowest connection or the monitor connection may be able to run with fetches spaced farther apart.

Discarded Fetch Data may be the root cause of Scan Timeouts or may be associated with prolonged Data Analysis. Data Analysis is slowed by noisy data. Two possible root causes are no averaging (seldom a problem) and saturating or nearly saturating the ADC. Saturation can cause problems by bringing the peaks buried in the noise up high enough to see. Data analysis will try to fit each of these peaks along with the saturated peaks. Consider using a limited mass range scan if you are driving your ADC to saturation. If you are not intentionally saturating, restore factory defaults and run Auto Tune to find a better EMBias setting.

Missed scans may be caused by either of the issues discussed above, but if your Service Log shows no evidence of either of these, they are most likely caused because you are requesting the data too slowly. Call GetScanData faster. Another reason this could happen is that you are incrementing your scan number before calling GetScanData. Do not change the returned scan number: the parameter is treated as "I last saw this scan. Please send me a different one." If you increment it, you will miss the next available scan.

Finally, fatal gauge errors (for example, a filament burned out) or gauge state transitions (SCAN or MASS SPEC button presses) might occur without any indication:

- Controller errors might be missed if more than one connection exists to a controller. That is, the caller who would first see the Controller errors would get an error return of CONTROLLER_ERRORS, but later callers would not. The later caller would only see the Controller Errors when polling via GetErrorCountAndQueue.
- Gauge state changes can be detected via interpretation of the Controller status registers (within the header information in GetScanData or GetHeaderData) or via polling GetGaugeState. These state changes could occur because of a gauge error or because someone pushed the buttons on the controller.

To be thorough, include code to poll both the controller errors and the gauge state in applications whether or not the application runs as a master or a monitor. If the gauge state changes from SCAN to another state, you may want to stop requesting scan data. Individual controller errors can be handled as appropriate.

## NOTES

Series 835

# Granville-Phillips® Series 835 Vacuum Quality Monitor™ Programmer's Reference Manual


***Customer Service / Technical Support***

**MKS, Granville-Phillips Division**
6450 Dry Creek Parkway
Longmont, CO 80503 USA

Phone: 1-303-652-4400 or 1-800-776-6543
FAX: 1-303-652-2844
Email: *gp-csr@mksinst.com*

**Corporate Office**
**MKS Instruments, Inc.**
2 Tech Drive, Suite 201
Andover, MA 01810 USA

Phone: 1-978-645-5500
www.mksinst.com