RP Metrix LASOMX.ocx

ActiveX Control interface for the RP Metrix LASOM1 (Laboratory Apparatus Scalable Olfactometer Module, Version 1).

The external entry points are described below.

Most functions return 0 if success, negative value if otherwise.
For many functions, the following parameters have common usages:

Parameter **nSlaveIndex** selects one LASOM module in a multi-slave system. Range is 1 to 15.
Use 1 for a single module system.

Parameter **nMfcIndex** selects one Mass Flow Controller on the selected slave module. Range is 1 to 2.

Parameter **bActivate** is non-zero to activate a valve or digital output.

Parameter **pnActive** points to a 32-bit signed integer used to receive the current state of a digital input.

Parameter **fDurationSec** causes an active digital output to be deactivated after the specified delay. If 0, the parameter is ignored.

long LASOMX.DevOpen(long nDeviceID, long nTestMode);

All sessions must call this function once, before any other functions are called.
This opens a connection to the LASOM USB device.
All valves are deactivated as a result of this call.

Parameter **nDeviceID** = 0, to open the first RP Metrix device found.
Each device contains a factory assigned 32-bit ID (use GetID() to read the value.)
Supply the factory assigned value to open that device specifically. This is only necessary if multiple RP Metrix USB devices are attached to the same computer.

Parameter **nTestMode** = 0 for normal operation.
= 1 for verbose operation (not implemented)
= 2 for demo mode without hardware (not implemented)

BSTR LASOMX.GetLastError();

Returns a string describing the most recent error. Example value: “No Device”.

BSTR LASOMX.GetID();

Returns a string identifying the LASOM unit and firmware in use.
Example value: “RP Metrix, LASOM1, Jul 05 2007 09:00:50 ID:2004”

long LASOMX.SetBlankState (long nSlaveIndex, long bVisible);

The valve indicator LEDs normally follow the activation of corresponding odor or gate valves.
LED blanking is an on/off function which can be used to hide all of these LEDs.
An independent mechanism using a DAC channel controls the brightness of these LEDs.

Parameter **bVisible** is non-zero to enable the valve indicator LEDs to be visible.

long LASOMX.SetOdorValve(long nSlaveIndex, long nOdorValveIndex, long bActivate);

Parameter **nOdorValveIndex** selects an odor valve. Range is 1 to 11. Odor valve 11 is reserved for the 'dummy' valve, which is active if no other odor valve is active to maintain consistent flow within a LASOM module. Previously activated odor valves will be automatically deactivated each time a new odor valve is activated. Deactivating an active non-dummy odor valve will automatically activate the dummy valve.

Parameter **bActivate** is non-zero to activate a valve.

long LASOMX.SetGateValve(long nSlaveIndex, long nGateValveIndex, long bActivate);

Parameter **nGateValveIndex** selects a gate valve. Range is 12 to 16. These valves are external to the LASOM module. Each may be operated independently.

Parameter **bActivate** is non-zero to activate a valve.

long LASOMX.SetGateValve2(long nSlaveIndex, long nGateValveIndex, long bActivate
, float fDurationSec);

Parameter **nGateValveIndex** selects a gate valve. Range is 12 to 16. These valves are external to the LASOM module. Each may be operated independently.

Parameter **bActivate** is non-zero to activate a valve.

long LASOMX.SetDigOutMask(long nOutputMask);

(TBD) Set the new output state of digital output pins on the master LASOM module.

Parameter **uOutputMask** activates digital outputs as described in "LASOM_DigMasks.txt". Set specific bits in this 32-bit parameter to activate the corresponding selected outputs.

long LASOMX.GetDigInMask(void);

Read the current input state of digital input pins on the master LASOM module.

The return value is a 32-bit unsigned integer which contains the sampled state of all digital input pins as described in "LASOM_DigMasks.txt". AND the result with a 32-bit mask to select particular digital inputs to observe.

long LASOMX.ArmGateValves(long nSlaveIndex, long nValveMask);

Prepare to activate selected gate valves on a future Strobe signal.

Parameter **uValveMask** selects multiple gate valves to be activated. Set bits in the mask based on the selected gate valve index of each valve to activate. Each mask bit is computed as 2^(GateValveIndex-1), so that for example the mask bit for valve 16 is 0x8000. Set multiple bits to activate multiple valves. For example, to active gate valves 12 and 16, use the mask value 34816 = 0x8800 = (2^(16-1)) + (2^(12-1)).

long LASOMX.SetStrobeMask(long nStrobeMask);

Set the Strobe mask, which either generates a software Strobe signal, enables an external Strobe input, or both.

Parameter **uStrobeMask** enables Strobe inputs, outputs, and internal strobe generation as described in "LASOM_DigMasks.txt".

If gate valves have been armed and the software Strobe signal is activated, the valves activate immediately when this function is called.

If gate valves have been armed and the Strobe In BNC input is enabled with this function, the valves will not activate until a TTL high level is present at the Strobe In input.

long LASOMX.SetMfcFlowRate(long nSlaveIndex
, long nMfcIndex
, float fPercentOfCapacity
);

Set the MFC mass flow rate set point.

Parameter **fPercentOfCapacity** requests a flow as a percentage of total capacity. Range is 0.0 to 100.0.

long LASOMX.GetMfcCapacity(long nSlaveIndex
, long nMfcIndex
, float FAR* pfCapacity
, BSTR FAR* pstrUnits
);

Inquire the MFC to determine the total flow rate capacity and the physical units in which the capacity is expressed.

Parameter **pfCapacity** points to a single precision float variable to receive the capacity value. Example values: 100.0 or 1000.0.

Parameter **pstrUnits** is a pointer to a character string. This string variable will be modified to contain the length and text of the capacity physical unit string. Example value: “sccm”.

long LASOMX.GetMfcFlowRateSetting(long nSlaveIndex
, long nMfcIndex
, float FAR* pfPercentOfCapacity
);
float LASOMX.GetMfcFlowRateSetting2(long nSlaveIndex
, long nMfcIndex
);

Get the current mass flow rate setting as a percentage of total capacity. Expected range is 0.0 to 100.0.

Parameter **pfPercentOfCapacity** points to a single precision float variable to receive the current setting.

long LASOMX.GetMfcFlowRateMeasure(long nSlaveIndex
, long nMfcIndex
, float FAR* pfPercentOfCapacity
);
float LASOMX.GetMfcFlowRateMeasure2(long nSlaveIndex
, long nMfcIndex
);

Get the current mass flow rate measurement as a percentage of total capacity.

Parameter **pfPercentOfCapacity** points to a single precision float variable to receive the current measurement. Expected range is 0.0 to 100.0.

The flow rate information is also maintained as a percentage of total capacity in the properties:

float	**MfcFlowRateSettingPercent**	**Current setting, range 0.0 to 100.0**
float	**MfcFlowRateMeasurePercent**	**Current measurement, range 0.0 to 100.0**

Gets can be issued for these property values, with the functions:

float LASOMX.GetMfcFlowRateSettingPercent (long nSlaveIndex, long nMfcIndex);
float LASOMX.GetMfcFlowRateMeasurePercent (long nSlaveIndex, long nMfcIndex);

A Put can be issued for the setting property, with the function:

float LASOMX.SetMfcFlowRateSettingPercent (long nSlaveIndex, long nMfcIndex, float fNewValue);

Some state information is maintained in the properties:

long	**Active**	**Result of GetBeam(), etc.**
long	**DigInMask**	**Sampled digital inputs at master.**
BOOL	**Beam(long nIndex)**	**Beam input k, where k = 1..3**
BOOL	**DigIn(long nIndex)**	**Digital input k, where k = 1..3**
BOOL	**XLogicIn(long nIndex)**	**XLogic input k, where k = 1..4**

Gets can be issued for these property values, with the functions:

long LASOMX.GetActive(void);
BOOL LASOMX.GetBeam(long nIndex);
BOOL LASOMX.GetDigIn(long nIndex);
BOOL LASOMX.GetXLogicIn(long nIndex);

long LASOMX.SetCue(long nIndex
, long bActivate
, float fDurationSec
);
Set a Cue output state on Leaf Board 1 (via J51).
Parameter **nIndex** selects the output. Range 1..3 corresponds to leaf J44..J46.

long LASOMX.SetDigOut(long nIndex
, long bActivate
, float fDurationSec
);
Set a DigOut output state on Leaf Board 2 (via J52).
Parameter **nIndex** selects the output. Range 1..3 corresponds to leaf J44..J46.

long LASOMX.SetXLogicOut(long nIndex
, long bActivate
, float fDurationSec
);
Set an XLogic output state (via J53).
Parameter **nIndex** selects the output. Range 1..4 correspond to Xlogic 5..8.

long LASOMX.GetBeam(long nIndex);

Get a Beam input state from Leaf Board 1 (via J51).
Parameter **nIndex** selects the input. Range 1..3 corresponds to leaf J41..J43.
Set the **Active** property with the result.

long LASOMX.GetBeam2(long nIndex
, long* pnActive
);
Get a Beam input state from Leaf Board 1 (via J51).
Parameter **nIndex** selects the input. Range 1..3 corresponds to leaf J41..J43.

BOOL LASOMX.GetDigIn(long nIndex);

Get a DigIn input state from Leaf Board 2 (via J52).
Parameter **nIndex** selects the input. Range 1..3 corresponds to leaf J41..J43.

BOOL LASOMX.GetXLogicIn(long nIndex);

Get an XLogic input state (via J53).
Parameter **nIndex** selects the input. Range 1..4 corresponds to Xlogic 1..4.

long LASOMX.MacroEnable();

Enable automatic use of LASOM sequencer to perform operations with repeatable timing. Normally, operations are initiated individually from the host, with host dependent timing. The following operations are affected:

SetOdorValve
SetGateValve
SetGateValve2
SetCue
SetDigOut
SetXLogicOut

long LASOMX.MacroDisable();

Disable automatic use of LASOM sequencer.

long LASOMX.ParseSeqFile(LPCTSTR pszFileName);

Read and parse a LASOM sequencer source file (LSQ file). The parameterized sequence will be stored in host memory, awaiting further action. The parser stops on the first line containing an error. See **GetSeqParseResult()** and **GetSeqNumParsedLines()**.

BSTR LASOMX.GetSeqParseResult(void);

Returns a string describing the success or failure of **ParseSeqFile()**.

long LASOMX.GetSeqNumParsedLines(void);

Returns the number of lines processed by **ParseSeqFile()**. This indicates the line on which an error stopped the parser, or the total number of lines if the parser succeeded.

long LASOMX.SetParamValue(LPCTSTR pszName, long nValue);

Define a sequencer parameter and assign the value. This function is used to override default parameter values stored within an LSQ file. All override assignments must occur BEFORE calling ParseSeqFile to have any effect. An error occurs if the parameter is already defined.

long LASOMX.ClearSequence();

Delete any previously defined sequencer parameter definitions resulting from parsing an LSQ file or setting override parameter values.

long LASOMX.CompileSequence();

Convert the parsed sequence to LASOM sequencer instructions. All parameter substitution and conditional and repeat generation takes place during this operation.

long LASOMX.LoadAndRunSequencer(long bClearStates);

Load the compiled sequencer instructions into the LASOM and start execution.

Parameter **bClearStates** must be zero to preserve the sequencer states from a previous execution.

long LASOMX.SetSequencerVar(long nVarIndex, long nNewValue);

Change the value of a sequencer variable.

Parameter **nVarIndex** is in the range 1..8.

long LASOMX.GetSequencerVar(long nVarIndex);

Get the value of a sequencer variable.

Parameter **nVarIndex** is in the range 1..8.

Sequencer Status Updates

The sequencer action "EmitStatus" causes a status update message to be emitted from the LASOM master to the host computer. LASOMX maintains a copy of the most recent status message. Status updates also occur when the sequencer is initialized and when it drops to the idle state.

The status information is maintained in the properties:

long	**SeqUpdateEnable**	**1 if Running, 0 if Idle**
long	**SeqUpdateStep**	**Current sequencer instruction index**
long	**SeqUpdateCount**	**Increments mod 256 on each update**
long	**SeqUpdateInState**	**16-bit digital input mask**
long	**SeqUpdateOutState**	**16-bit digital output mask**
long	**SeqUpdateVarState (long nVarIndex)**	**$Vark, where k = nVarIndex = 1..8**

Gets can be issued for these property values, with the functions:

long LASOMX.GetSeqUpdateEnable(void);
long LASOMX.GetSeqUpdateStep(void);
long LASOMX.GetSeqUpdateCount(void);
long LASOMX.GetSeqUpdateInState(void);
long LASOMX.GetSeqUpdateOutState(void);
long LASOMX.GetSeqUpdateVarState(long nVarIndex);

Parameter **nVarIndex** is in the range 1..8.

Use **GetSeqUpdateCount()** to check for the arrival of a new status message. When it differs from the previous call, a new message has arrived and the other status properties may have changed.

Sequencer Execution Timing

In addition to the use of timers to record event times, delays, and durations the sequencer maintains a record of the last execution time of every instruction. Although the mapping of a source code statement to the corresponding instruction offset is determined by the compiler, a user can access the time for any number of specific steps in a sequencer program which are prefixed with Label statements. The Label statement itself consumes no instruction space in the sequencer. The relative time in seconds from execution of the first sequencer instruction can be obtained by either of the following entry points:

long LASOMX.GetSequencerLabelTime(LPCTSTR lpszLabel, float FAR* pfSeconds);
float LASOMX.GetSequencerLabelTimeValue(LPCSTR lpszLabel);

Parameter **lpszLabel** is a string which should match a label in the source code, such as "@EndLoop1".
Parameter **pfSeconds** is a pointer to a float variable in which the relative time in seconds is returned.
The return value of **GetSequencerLabelTime** indicates the success of the operation, and if the label was recognized.
The return value of **GetSequencerLabelTimeValue** is the relative time in seconds. The value is undefined if the label is not recognized.

long LASOMX.GetSequencerTimer(long nIndex, float FAR* pfSeconds);

Get the absolute time value of a sequencer timer in seconds.

Parameter **nIndex** is in the range 0..7.

LASOM Analog Input and Output

LASOM connector J33 as described in "LASOM_Analog.txt" provides access to the on board 4-channel ADC/DAC interface chip (Analog Devices ADT7517). This device supports analog control of the MFC modules, and adjustable dimming of the valve indicator LEDs. If those functions are not needed, the connector provides for other external uses.

DAC channels 1 to 4 are available on connector J33.
ADC channels 1 to 4 are available on connector J33.
ADC channel 5 measures the chip temperature in degrees C (not volts).
ADC channel 6 measures the chip power supply input, nominally 5 volts.

```
long LASOMX.SetDac(long nSlaveIndex
                   , long nDacIndex
                   , float fVoltage
                   );
```

Set the DAC output voltage.

Parameter **nDacIndex** range is 1 to 4.
Parameter **fVoltage** range is 0.0 to 5.0.

```
long LASOMX.GetAdcMeasure(long nSlaveIndex
                   , long nAdcIndex
                   , float FAR* pfVoltage
                   );
float LASOMX.GetAdcMeasure2(long nSlaveIndex
                   , long nAdcIndex
                   );
```

Get the current ADC measurement.

Parameter **nAdcIndex** range is 1 to 6.
Parameter **pfVoltage** points to a single precision float variable to receive the current measurement. Expected range is 0.0 to 2.5 volts for channels 1 to 4.