LabMax™ User Manual Addendum

This addendum modifies information originally published in the following documents:

- *LabMax-TOP User Manual* (part number 1143715)
- *LabMax-TO User Manual* (part number 1143716)

Topics discussed in this addendum include:

- Battery directive (this page)
- Dip switch (OEM mode) (page 2)
- ActiveX (page 2)
- Pulsed thermopile joules mode (page 13)
- Collecting pulsed thermopile joules data over a host port (page 13)
- Thermopile sensors, long pulse joules (page 14)

Battery Directive

The batteries used in this product are in compliance with the EU Directive 2006/66/EC ("EU Battery Directive").

Batteries Contained in this Product

Description	Type
4400 mAH battery pack	Lithium Ion

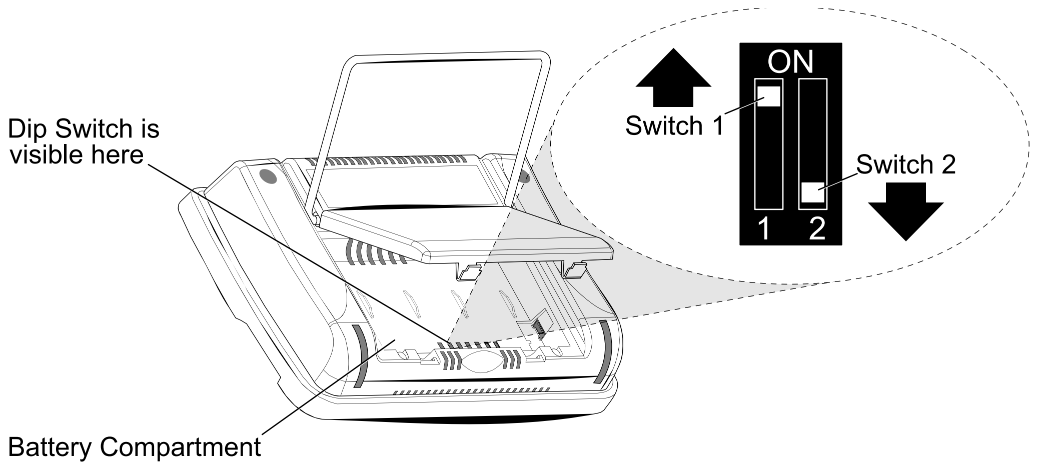
Dispose of batteries according to local regulations. Do not dispose as normal waste. Consult your local waste authorities for guidance.

Dip Switch (OEM Mode)

Setting the Dip switch to OEM mode ensures that if power is lost while the meter is in use, it will automatically turn back on again when power is restored. The benefit of this mode is that, following a power loss, remote installations can communicate with the meter without having to manually recycle instrument power.

To activate OEM mode, move Switch 1 on the Dip switch to the ON position (see the following figure).

To avoid prematurely draining the onboard battery (which could result in erratic meter readings), leave Switch 2 in the down (OFF) position.



ActiveX

Document Overview

This section specifies the ActiveX control interface for the Coherent LabMax single-channel meter. The purpose is to detail the properties, events, and methods of the control so that it can easily integrate into a variety of containers such as LabVIEW, Visual Basic, VB.Net, and C#.

Scope

This section defines the interface for the LabMax low-level ActiveX control. The actual high-level commands, responses, and behavior that you can expect from the meter resulting from these interface calls are in the *Host Interface* section of the user manual.

Installation

The ActiveX control and low-level DDL are automatically installed—along with the LabMax PC Applications software—when you run the CD that is supplied with the meter.

LabMax Low Level Control

Overview

The LabMax Low Level Control (LabMaxLowLevCtl.ocx) provides a simple interface that enables a container to access all interfacing capabilities available through the host interface. This control hides the differences relating to specific details when communicating with the meter through USB, RS-232 or GPIB. Only one instance of the control is allowed per process and only a single mode of communication (USB, RS-232 or GPIB) is allowed per control. In other words, the communication modes are not mixed per process.

Before using any methods or properties in the control, you must call the Initialize() method. This method sets up the necessary member variables to enable proper connection to the meter(s). Likewise, when the connection is no longer needed, you must call the DeInitialize() method. The DeInitialize disconnects from any attached meters and enables the caller to call Initialize() again without destroying the control. Note that calling DeInitialize() to disconnect meters will not cause the MeterRemoved event to fire. If you want to receive this event, call DisconnectFromMeter() before calling DeInitialize().

The command set is based on the SCPI Standard. A major deficiency of SCPI is that commands and queries do not have defined replies and illegal commands and queries are silently ignored, except for standard error queuing. To have a more robust method of communication, this control establishes a command/query handshaking protocol, which allows it to immediately identify an illegal command/query and reflect this through the interface. For more information, see the specification for “SendCommandOrQuery” on page 8.

To support the command/query and reply handshaking protocol, a queue is implemented that contains strings received from the meter after a command/query has been sent. This queue is flushed if you send another command/query to the meter. It is up to you to know how many calls to GetNextString are needed to retrieve the strings in the queue, and that the data in the queue is lost if you initiate another command/query without retrieving the data from the previous query.

Quick Reference

Below is a summary of the properties, methods, and events exposed in the control.

	PURPOSE	PAGE
PROPERTIES		
CommunicationMode(…)	Specifies whether USB, RS-232 or GPIB is used	5
RS232Settings(…)	Specifies port settings for a specific meter	6
GPIBSettings(…)	Specifies board index for the GPIB board	7
SerialNumber(…)	Retrieves serial number for a specific meter	7
METHODS		
Initialize()	Initialize the control	8
DeInitialize()	Deinitializes the control; disconnects all meters	8
SendCommandOrQuery(…)	Sends specified SCPI command or query to meter	8
GetNextString(…)	Returns string received from meter	9
ConnectToMeter(…)	Connects to a specific meter	10
DisconnectFromMeter(…)	Disconnects from a specific meter	10
EVENTS		
MeterAdded(…)	Meter has established a communication channel	10
MeterRemoved(…)	Meter has broken a communication channel	11
AsynchronousNotification(…)	An asynchronous notification event is transmitted from the meter	11
USBStreamingPacket(…)	A streaming measurement data packet is available	12

Detailed Reference

Below is detailed information on the properties, methods and events exposed in the control. Some of the properties and all of the methods and events have a parameter called *iMeterIndex*. The *iMeterIndex* parameter contains a value specific to the communication mode setup by the **CommunicationMode** property. The following values are valid for the *iMeterIndex* parameter:

USB CommunicationMode

Valid inputs are 0 to 127. The control automatically detects any USB devices, either when the communication mode is set to COM_MODE_USB, or when a device is plugged in. The *iMeter-Index* is assigned by the control and is sent to the MeterAdded () event. Thereafter, this parameter should be used in the calls to the methods and properties of the control.

GPIB CommunicationMode

Valid inputs are 1 to 31. Address 0 is reserved for the GPIB controller-in-charge board. The assigned number must match whatever is currently set in the meter hardware.

RS232 CommunicationMode

Valid inputs are 1 to 9, depending on which COM port the meter is connected.

<u>Properties</u>

CommunicationMode

The *CommunicationMode* property defines the single mode of communication for this instance of the ActiveX control. If a communication channel is connected and you try to reset this mode, the call is ignored. The control is inactive until this property is set. The property is defined by the enumerated type *LabMaxCommunicationMode*, defined as follows:

```
typedef enum {
        COM_MODE_NONE,       // 0
        COM_MODE_USB,        // 1
        COM_MODE_GPIB,       // 2
        COM_MODE_RS2322      // 3
} LabMaxCommunicationMode;
```

The initial value for the property is COM_MODE_NONE.

HRESULT CommunicationMode
([in] LabMaxCommunicationMode
iCommunicationMode)

Inputs:

iCommunicationMode—value in the enumerated type *LabMaxCommunicationMode*, specified above.

HRESULT CommunicationMode
([out, retval] LabMaxCommunicationMode
***oCommunicationMode)**

Outputs:

oCommunicationMode—value in the enumerated type *LabMaxCommunicationMode*, specified above.

RS232Settings

The *RS232Settings* property specifies the serial port settings of the addressed meter. It uses the same specification format as the MSCOMM32 control settings property. Basically, it is a string composed of four settings and has the following format: “B,P,D,S” where B is the baud rate, P is the parity, D is the number of data bits, and S is the number of stop bits. The default value is “19200,N,8,1”.

- Valid baud rates are: 9600, 19200(default), 38400, 57600, and 115200.
- Valid parity values are: E (even), N (none—default), and O (odd).
- Valid data bit value is: 8(default)
- Valid stop bits values are: 1(default), and 2.

```
HRESULT RS232Settings
            ([in] SHORT iMeterIndex,
            [out,retval] BSTR* oRS232Settings)
```

Inputs:

iMeterIndex—valid parameter values depend on the communication mode. For more information, refer to “Detailed Reference” on page 4.

Outputs:

oRS232Settings—a BSTR holding the serial port settings string, as defined above.

```
HRESULT RS232Settings
            ([in] SHORT iMeterIndex,
            [in] BSTR    iRS232Settings)
```

Inputs:

iMeterIndex—valid parameter values depend on the communication mode. For more information, refer to “Detailed Reference” on page 4.

iRS232Settings—a BSTR holding the serial port settings string, as defined above.

Because this property takes an input parameter, it can appear differently when used by VB.Net and LabVIEW. For example, in VB.Net, the property is accessed by *set_RS232Settings(...)* and *get_RS232Settings(...)*. For LabVIEW, the property appears under the Methods menu as *RS232Settings(put)* and *RS232Settings(get)*.

GPIBSettings

The *GPIBSettings* property specifies the board index for the GPIB board. The default value is “0”. Only one board index is supported for this instance of the ActiveX control. All referenced meters must be attached to this board.

HRESULT GPIBSettings
([out,retval] BSTR* oGPIBSettings)

Outputs:

oGPIBSettings—a BSTR holding the GPIB board index string.

HRESULT GPIBSettings
([in] BSTR iGPIBSettings)

Inputs:

iGPIBSettings—a BSTR holding the GPIB board index string.

SerialNumber

The *SerialNumber* property indicates the serial number of the addressed meter.

HRESULT SerialNumber
(in] SHORT iMeterIndex,
[out,retval] BSTR* oSerialNumber)

Inputs:

iMeterIndex—valid parameter values depend on the communication mode. For more information, refer to “Detailed Reference” on page 4.

Outputs:

oSerialNumber—a BSTR holding the serial number of the addressed meter. It will be an empty string if an invalid meter is addressed.

Because this property takes an input parameter, it can appear differently when used by VB.Net and LabVIEW. For example, in VB.Net, the property is accessed by *get_SerialNumber(...)*. In LabVIEW, the property appears under the Methods menu as *SerialNumber*.

Methods

Initialize

You must call the *Initialize* method before calling any other methods. This is equivalent to a C++ constructor and is used to initialize the control variables.

HRESULT Initialize
([out, retval] SHORT* oResult)

Outputs:

oResult—the method returns 1 if the initialization was successful; otherwise, 0.

DeInitialize

You must call the *DeInitialize* method when communication with the control is no longer needed. This is equivalent to a C++ destructor. It ensures that all meters are properly disconnected and allocated memory is freed. If you use this method—as opposed to *DisconnectFromMeter*—to handle disconnecting from all connected meters, the *MeterRemoved* event will not be fired.

HRESULT DeInitialize
([out, retval] SHORT* oResult)

Outputs:

oResult—the method returns 1 if the initialization was successful; otherwise, 0.

SendCommandOrQuery

The *SendCommandOrQuery* method allows the container to send a specified SCPI command or query to the addressed meter specified by the parameter **iMeterIndex**. This method waits for a response from the meter to state whether the command/query is valid or not. The timeout value for this response is 3 seconds. After issuing the command/query, it is expected that you call *GetNextString()* to retrieve the string(s) in response to this action.

HRESULT SendCommandOrQuer
([in] SHORT iMeterIndex,
[in] BSTR iCommandOrQuery,
[out,retval] SHORT* oResult)

Inputs:

iMeterIndex—valid parameter values depend on the communication mode. For more information, refer to “Detailed Reference” on page 4.

iCommandOrQuery—contains a user-specified SCPI command or query string.

Outputs:

oResult—the method returns a 1 if the string is successfully transmitted and the validation of the command/query from the meter is received before the timeout period. A 0 is returned if the timeout expires, the command/query is invalid, an invalid meter is specified as an input parameter, or the *CommunicationMode* property is not set.

GetNextString

The *GetNextString* method allows the caller to retrieve the next available string in the string queue of the addressed meter. This queue is filled according to the specific command/query sent to the meter in the *SendCommandOrQuery()* method. Refer to the “Send-CommandOrQuary” heading, above, for more information.

HRESULT GetNextString
([in] SHORT iMeterIndex,
[out,retval] BSTR* oString)

Inputs:

iMeterIndex—valid parameter values depend on the communication mode. For more information, refer to “Detailed Reference” on page 4.

Outputs:

oString—the method returns the next available string in the string queue of the addressed meter. An empty string is returned if there is no string available in the queue, if an invalid meter is addressed, or if the CommunicationMode property is not set.

ConnectToMeter

The *ConnectToMeter* method connects to the addressed meter if the communication mode is RS-232 or GPIB. If the communication mode is RS-232, it uses the settings as defined by the *RS232Settings* property. If the communication mode is USB, this method does nothing.

```
HRESULT ConnectToMeter
            ([in] SHORT iMeterIndex,
            [out,retval] SHORT* oResult)
```

Inputs:

iMeterIndex—valid parameter values depend on the communication mode. For more information, refer to “Detailed Reference” on page 4.

Outputs:

oResult—this method returns 1 if the communication mode is RS-232 or GPIB, and a successful connection is established. If the communication mode is USB, the method returns 1; otherwise, it returns a 0.

DisconnectFromMeter

The *DisconnectFromMeter* method disconnects from the addressed meter if the communication mode is RS-232 or GPIB. If the communication mode is USB, this method does nothing.

```
HRESULT DisconnectFromMeter
            ([in] SHORT iMeterIndex,
            [out,retval] SHORT* oResult)
```

Inputs:

iMeterIndex—valid parameter values depend on the communication mode. For more information, refer to “Detailed Reference” on page 4.

oResult—this method returns 1 if the communication mode is RS-232 or GPIB, and connection is successfully broken. If the communication mode is USB, the method return 1; otherwise, it returns a 0.

Events

MeterAdded

The *MeterAdded* event fires when a meter establishes a connection. If the communication mode is RS-232 or GPIB, this event occurs after a successful call to *ConnectToMeter()*. If the communication mode is USB and a meter is already plugged in a USB port, this event will occur immediately after setting the control in communication mode to USB. If the communication mode is USB and a meter is plugged in after the communication mode is configured, this event will occur when the plug-in is detected. For USB, the **iMeterIndex** is assigned by the control and should be used in further calls to methods and properties relating to this device. For RS-232 and GPIB, the **iMeterIndex** is the index that is sent in the *ConnectToMeter()* function call.

void MeterAdded
([in] SHORT iMeterIndex)

Inputs:

iMeterIndex—valid parameter values depend on the communication mode. For more information, refer to “Detailed Reference” on page 4.

MeterRemoved

The *MeterRemoved* event fires when the connection with the configured meter is broker. If the communication mode is RS-232 or GPIB, this event occurs after a successful call to *DisconnectToMeter()*. If the communication mode is USB, this event occurs when a meter is unplugged from the USB port.

void MeterRemoved
([in] SHORT iMeterIndex)

Inputs:

iMeterIndex—valid parameter values depend on the communication mode. For more information, refer to “Detailed Reference” on page 4.

AsynchronousNotification

The AsynchronousNotification event fires when an asynchronous event is received from the meter. The communication mode determines when this event happens. For USB, it occurs when an attention packet is transmitted. For GPIB, it occurs when a service request is generated. For RS-232, it occurs when a service request is transmitted.

void AsynchronousNotification
([in] SHORT iMeterIndex)

Inputs:

iMeterIndex—valid parameter values depend on the communication mode. For more information, refer to “Detailed Reference” on page 4.

USBStreamingPacket

The *USBStreamingPacket* event fires when a streaming measurement data packet is generated by the meter. This event only occurs if the communication mode is USB.

void USBStreamingPacket
([in] SHORT iMeterIndex,
[in] VARIANT iStreamingPacket)

Inputs:

iMeterIndex—valid parameter values depend on the communication mode. For more information, refer to "Detailed Reference" on page 4.

iStreamingPacket—the VARIANT data type is an array of VARIANTS. Each array element is a VARIANT, and maps to the data type and value of each structure element.

The C data structure is:

```
typedef struct LMHostMsg {
        float measurement;
        float quadDistanceMMX;
        float quadDistanceMMY;
        unsigned char displayFormatId;
        unsigned char scaleMultiplierId;
        unsigned char unitsId;
        unsigned char flags;
        unsigned int pulsePeriodUS;
        unsigned int sequenceNumber;
};
```

The corresponding VARIANT array is:

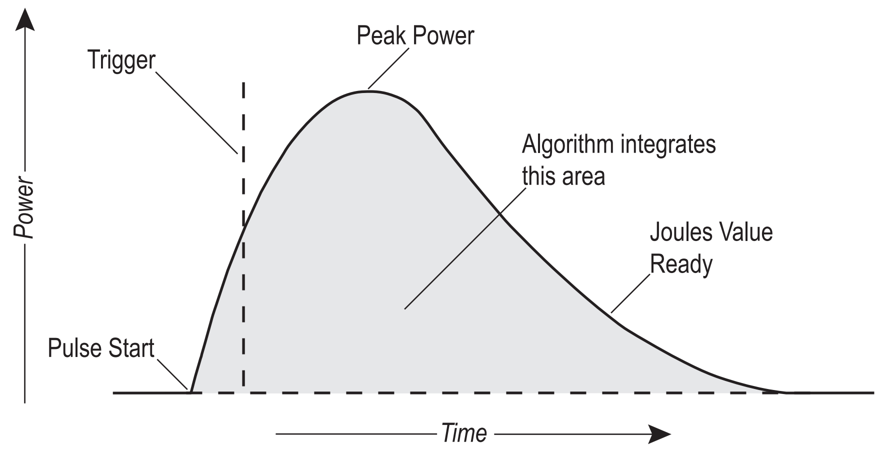
array [0]—4-byte float (in Labview, "Single Precision")
array [1]—4-byte float
array [2]—4-byte float
array [3]—1-byte unsigned (in Labview, "Unsigned Byte")
array [4]—1-byte unsigned
array [5]—1-byte unsigned
array [6]—1-byte unsigned
array [7]—4-byte unsigned (in Labview, "Unsigned Long")
array [8]—4-byte unsigned

Pulsed Thermopile Joules Mode

(*for long-pulsed lasers only*) When a thermopile sensor is attached, the meter has the capability of measuring energy from a finite duration laser pulse, or from a series of finite duration laser pulses. (Thermopile sensors are typically used to measure laser power and have an extremely slow response time relative to the pulse width of the laser used to generate the power signal.)

The power curve (shown below) is integrated from the pulse start to infinity. The final energy value is algorithmically calculated shortly after peak power is attained.

The trigger is *not* determined by the power threshold, but by the slope of the change in power.



Collecting Pulsed Thermopile Joules Data Over a Host Port

To collect pulsed thermopile joules data:

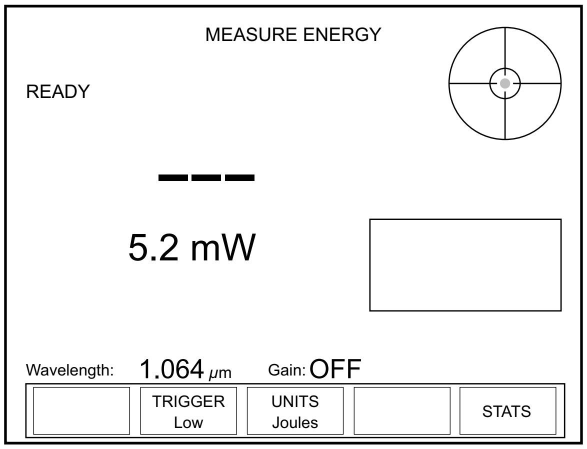
1. Set the mode to Joules by using either the UNITS soft key on the meter, or the CONFigure:MEASure host command.
2. Query the host for measurement data by using the FETCh:NRECords query. (For detailed information, refer to “Data Query” in the *Host Interface* section of the user manual.)

The type of data that will be collected depends entirely on which mode is currently selected: Joules mode will collect only joules data and Watts mode will collect only watts data.

Thermopile Sensors, Long Pulse Joules

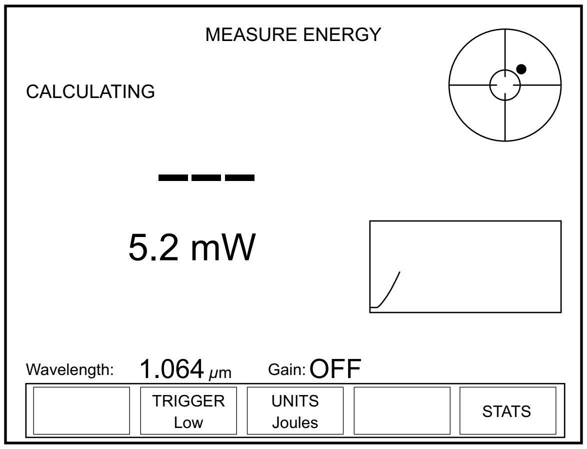
Connecting a thermopile sensor to the meter and switching to Joules mode displays the screen shown below**.** This mode measures the energy of single pulses between 1 millisecond and 10 seconds in length, and with energies from 5 mJ to hundreds of joules. Measuring long mJ pulses requires more sensitive thermopiles (such as a PS10Q)**.**

When measuring energy with a thermopile sensor, the energy range is sensor-dependent and is automatically determined by the meter. The range value cannot be manually changed.



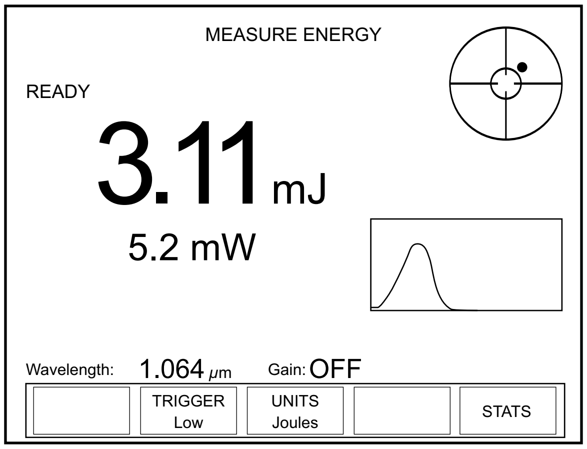
MEASURE ENERGY: Thermopile Sensors (Long Pulse Joules)

Display Smoothing and Speedup are not active in this mode. When a pulse is received, the READY indicator in the upper left of the screen changes to CALCULATING, the position indicator shows the beam position in the crosshairs of the bulls eye target (quad sensors only), and a plot of the thermopile response begins, as shown below.



MEASURE ENERGY: Thermopile Sensors (Long Pulse - Active)

Once the calculation has been completed, the new Joules value is displayed, the CALCULATING indicator is replaced with READY, and the plot stops (see the following figure). At this point, the meter is ready to receive another pulse.



MEASURE ENERGY: Thermopile Sensors (Long Pulse - After Pulse)

Soft Keys

There are three soft keys on this screen: TRIGGER, UNITS, and STATS.

TRIGGER

- Available values: Low (factory default), Medium, and High.

The purpose of the TRIGGER is to filter out ambient noise. Low—the most sensitive of the three values—does the least filtering. If that value produces an excessive number of false triggers while taking a reading, select the Medium or High value to increase the filtering.

Pressing the TRIGGER soft key cycles through the three values.

UNITS

Pressing the UNITS soft key toggles the measurement units back to Watts and displays the MEASURE POWER screen.

STATS

Pressing the STATS soft key displays the STATS screen. For details on the STATS screen, refer to information under “Statistics” in the Operation section of the User manual.

Questions?

Call:	(USA) 1.800.343.4912
	(Europe) +49-6071-968-0
	(International) 503.454.5700
or visit our website:	www.Coherent.com

LabMax™ User Manual Addendum
© Coherent, Inc., 3/2009, (RoHS). Printed in the U.S.A.
Part No. V006341