Ocean Optics

OmniDriver

Programming Guide

Document Number 000-20000-400-12-201505

For Products: OmniDriver and SPAM
Document: 000-40000-010-02-201505

Table of Contents

About This Manual

Document Purpose and Intended Audience

This document provides you with an installation and operating instructions for OmniDriver.

What's New in this Document

This version of the *OmniDriver Programming Guide* is updated for Release 2.38.

Document Summary

Chapter	Description
Chapter 1: *Introduction*	Provides an overview of the OmniDriver/SPAM software.
Chapter 2: *OmniDriver Installation*	Contains instructions for installing OmniDriver/SPAM on a Windows, Mac or Linux platform from the Ocean Optics website or from the OmniDriver software DVD.
Chapter 3: *Basic Sequence of Operations*	Describes the typical sequence of operations that the application must perform to control a spectrometer and acquire data.
Chapter 4: *Acquisition Parameters*	Presents a complete list of all available acquisition parameters.
Chapter 5: *Optional Features*	Lists and describes the optional capabilities offered by some spectrometers.
Chapter 6: *External Trigger Modes*	Lists and describes the trigger modes available on various spectrometers.
Chapter 7: *High-Speed Data Acquisition*	Provides the function call sequence for using the High-speed Data Acquisition mode.
Chapter 8: *Using OmniDriver with Multithreaded Applications*	Describes how to use OmniDriver with multi-threaded applications.
Chapter 9: *Developing Your OmniDriver Application*	Provides information on setting up the development environment for your specific programming language to develop OmniDriver applications.
Chapter 10: *Sample Programs*	Contains sample programs for C, C++, C#, Delphi, Java, LabVIEW and Visual Basic.

Chapter	Description
Chapter 11: *Redistributing OmniDriver with Your Application*	Contains information for redistributing the runtime version of OmniDriver.
Chapter 12: *Wrapper API Reference*	Provides detailed information about a few of the wrapper functions.
Chapter 13: *SPAM Library*	Contains information about the SPAM library.
Appendix A: *FAQs*	Lists frequently asked questions and answers.
Appendix B: *Upgrading Your Application for OmniDriver 1.6*	Describes how to upgrade C/C++, Visual Basic, and VB6 applications for OmniDriver 1.6.
Appendix C: *Programming the ARCoptix Spectrometer*	Provides information for programming the ARCoptix ANIR Series of Fourier Transform Spectrometers (FTS).
Appendix D: *Controlling Memory Size*	Provides information for controlling the memory size of Java and your application.
Appendix E: *Improving OmniDriver Performance on Windows*	Contains suggestions for improving the performance of OmniDriver on Windows systems.
Appendix F: *Using the Timeout Feature*	Provides instructions for installing the Timeout feature.
Appendix G: *Customized Branding of OmniDriver*	Describes how to change the OmniDriver software name and company to customize it for your own company.

Product-Related Documentation

You can access documentation for Ocean Optics products by visiting our website at **http://www.oceanoptics.com**. Select Support → Technical Documents, then choose the appropriate document from the available drop-down lists.

- Detailed instructions for OceanView Software is located at: **http://oceanoptics.com/wp-content/uploads/OceanViewIO.pdf**

Upgrades

Occasionally, you may find that you need Ocean Optics to make a change or an upgrade to your system. To facilitate these changes, you must first contact Customer Support and obtain a Return Merchandise Authorization (RMA) number. Please contact Ocean Optics for specific instructions when returning a product.

Chapter 1

Introduction

Overview

OmniDriver® is a powerful Software Developer's Kit (SDK) for Windows, Mac and Linux operating systems that allows you to easily write custom software solutions for your Ocean Optics USB spectrometers. Ocean Optics is the first and only provider in the optical sensing industry to offer this level of cross-platform compatibility.

OmniDriver provides an Application Programming Interface (API) that is the culmination of our best software driver packages and allows you to harness the power of high-speed data acquisition in a single cross-platform driver. Integrate OmniDriver into your own software application for complete control over USB spectrometers and devices in virtually any OS environment.

OmniDriver is sold as two separate but interrelated products:

- **OmniDriver**. This product gives you the functions you need to develop applications which communicate with and control Ocean Optics spectrometers. With this product you can connect to spectrometers, set acquisition parameters such as integration time, and acquire spectra.
- **SPAM** (Spectral Processing and Math). This add-on product gives you functions to perform spectral processing such as peak-finding, radiometric color analysis, etc. You can use the functions in the SPAM library to process spectral data coming from any USB spectrometer, including non-Ocean Optics models. However, normally you will also want the OmniDriver product, which simplifies the process of obtaining spectral data from Ocean Optics spectrometers. See the Spectral Processing And Math (SPAM) Module Programming Guide for more information (see *Product-Related Documentation*).

OmniDriver Is Cross-platform

OmniDriver was created in the Java environment and includes native libraries for select Windows, Macintosh and Linux operating systems. Using OmniDriver, you can develop robust applications to control multiple Ocean Optics USB spectrometers across these different operating systems. Ocean Optics is the first and only provider in the optical sensing industry to offer this level of cross-platform compatibility.

Multi-Language Support

You can develop OmniDriver-based applications in the following languages.

- "C"
- C++ (Microsoft Visual Studio)
- C#

- Borland Delphi/Pascal
- Java
- LabVIEW (Windows only, Version 7.1 or greater)
- Visual Basic and VBA (Visual Basic for Applications)

Multiple Interface Methods

- “C” style calling convention (all platforms)
- C++ (available with Microsoft Visual Studio/Windows, also with gcc/Linux)
- COM object interface (Windows only)
- .NET assembly interface

Applications written in Java are environment-independent; they can work across all operating systems. OmniDriver is not only platform-independent, but also spectrometer-independent; the same Java code works with all Ocean Optics USB spectrometers.

OmniDriver Features

OmniDriver offers the following features:

HighResTiming

Time stamping that is accurate to sub-microsecond performance; great for chemical kinetics and other applications that require complex time accountability.

LabView Support

OmniDriver provides drivers for LabVIEW (Version 7.1 or greater) to enable you to configure Ocean Optics spectrometers as real-time virtual instruments in National Instruments' LabVIEW graphical programming environment.

Support for Ocean Optics USB devices

- ADC1000-USB (including Deep Well)
- Apex
- Curie
- Flame
- HR2000
- HR2000+
- HR4000
- Jaz
- Maya2000
- Maya2000Pro
- MayaLSL
- MMS-Raman
- NIR256
- NIR512
- NIRQuest

- QE65000
- QE65 Pro
- QE *Pro*
- Spark
- STS
- Torus
- USB2000
- USB2000-FLG
- USB2000+
- USB4000
- Ventana

Note

OmniDriver does NOT support PCI or ISA products.

Operating System Support

- Windows:
  - Windows XP, Vista, Windows 7, Windows 8 and 8.1
  - 64-bit versions of XP, Vista, and Windows 7
- Mac:
  - OSX 10.5 or later
- Linux:
  - Many x86 distributions are supported
  - Kernel 2.4.20 and libusb 0.1.10 or later are required

Note

Although OmniDriver supports the Linux and Mac OS/X platforms in addition to Windows, this programming guide is written from the point of view of a Windows developer. Often the differences between platforms is merely superficial (for example, Linux has “shared object” (*.so) files, and the Mac has “*.dylib” files, whereas Windows has “dynamic link libraries” (*.dll)). Where differences between platforms are more significant, details are broken out into sections for each platform.

Measurement Corrections:

- Detector nonlinearity
- Electrical dark (automatic baseline removal)
- Stray light
- Boxcar smoothing (averaging across pixels)
- Averaging multiple scans

Control of the following extended functions (depending on spectrometer model):

- Strobe enable
- Thermo-electric cooling (TEC), and reading detector and PCB temperatures
- Gated fluorescence mode
- Analog output (4-20mA current output for LS-450 and soon the AOUT; voltage out for HR4000)
- Analog input for the supported spectrometers (voltage only)
- Digital (TTL) input/output (control of GPIO pins)
- Setting external trigger modes
- Reading out wavelength calibration
- Setting single and continuous strobe delays

OmniDriver Architecture

You can access OmniDriver functionality via two DLL files:

- **OmniDriver32.dll** (or **OmniDriver64.dll** on 64-bit systems) contains the functions which allow you to control all spectrometer settings and acquire spectra. For example, you can set the integration time, scans-to-average, enable electric dark correction, etc.
- **SPAM32.dll** (or **SPAM64.dll** on 64-bit systems) contains higher-level functions for analyzing the spectra. For example, this .dll provides functions to perform peak-finding, color space calculations, and energy output measurements.

These two files can be purchased separately (as OmniDriver and SPAM), or together (OMNI+SPAM).

There is a third DLL file, common32.dll, which contains functions used internally by the other DLL files. These functions are not called directly by the user.

OmniDriver is Written in Java

The bulk of OmniDriver's capabilities are implemented in the Java language. This is key to OmniDriver's ability to run on multiple platforms. The functions in the above-mentioned DLLs merely invoke corresponding Java functions. These functions in the DLLs then translate the parameters of each function into the form expected by Java, and then invoke the underlying Java method. As a consequence, Java developers can ignore the DLLs and access the Java objects and methods directly from the OmniDriver.jar and SPAM.jar files.

OmniDriver comes bundled with the Java 1.6 Run-time Engine (JRE). Therefore, you do not need to install Java on your PC unless you are developing applications in the C/C++ or Java languages. In this case you will need to install the Java 1.6 JDK, which contains header files required by C/C++.

LabVIEW Development

For LabVIEW developers, OmniDriver provides a set of LLB/VI files which expose OmniDriver's functionality in a fashion that is natural to the LabVIEW development environment. Behind the scenes, these LLB/VI's invoke the methods contained in the DLL files that comprise OmniDriver.

USB Port Access

Most Ocean Optics spectrometers communicate with a computer via the USB port. One notable exception is the Jaz unit, which can communicate not only via USB, but also via Ethernet. To access the USB ports, we provide an additional DLL named NatUSB.dll. The methods in this DLL are invoked internally from the Java implementation of OmniDriver. The user should never invoke any of the methods in NatUSB.dll directly.

Wrapper Object

OmniDriver contains classes which represent each distinct Ocean Optics spectrometer (e.g., USB2000+, HR4000, etc.). However, all access to your spectrometers, regardless of model, should be through the Wrapper object. Your application should instantiate the Wrapper object and then use the methods of this Wrapper object to connect to and control your spectrometer. The methods belonging to the Wrapper object are collectively called the **Wrapper API**.

Chapter 2

OmniDriver Installation

Overview

OmniDriver can be downloaded from the Ocean Optics website's Software Downloads page, or retrieved from the CD that you received with your purchase of the software. In either case, you will need the password to complete the installation process. If you have a CD with the OmniDriver software, the password is located on the back of the CD jacket.

This chapter contains instructions for installing OmniDriver on each of the following operating systems:

- Microsoft Windows – XP, Vista, 7, 8, 8.1; 32-bit and 64-bit
- Mac – OS X version 10.5 or later on Intel processor
- Linux – Any version released for an x86 or amd64 platform since 2010

  Example: CentOS(Version 5.5), and Ubuntu (version 10.4LTS)

All sample applications are now located in a separate installer that can be downloaded from our website: **http://www.oceanoptics.com/technical/softwaredownloads.asp**. By default, samples are installed to C:\Ocean Optics\OmniDriverSamplePack – *version number*.

Note

After you have installed your OmniDriver software, please refer to *Installing the Spectrometer Driver Software*.

When the installation process is finished, the following subdirectories will be created beneath the OmniDriver "home" directory:

Subdirectory	Contents
docs	Main documentation area
docs\javadocs.omnidriver	Javadocs for OmniDriver library functions
docs\javadocs.spam	Javadocs for SPAM library functions
docs\javadocs.highrestiming	Javadocs for High Resolution Timing functions
docs\javadocs\ooiutils	Javadocs for utility functions

Subdirectory	Contents
_jvm	Java Runtime Engine (JRE) used by OmniDriver
include	Header files for use with C/C++ application development
LabVIEW	*.llb libraries and *.vi files to make the OmniDriver functions available to LabVIEW applications
OOI_HOME	The DLL, LIB, and JAR files which make up the OmniDriver and SPAM libraries This directory also contains the Linux *.so files and the MacOSX *.dylib files.
VisualBasic6_Declarations	**OmniDriver.bas** file for inclusion in VB6 applications that need to use the "C" interface to the functions in the OmniDriver DLL.
ezusb_driver	Windows USB drivers (for 32-bit platforms only)
winusb_driver	Windows USB drivers (for 64-bit platforms only)

Once you have installed the software, you'll want to verify your installation, look at the javadocs provided to get an idea of how the objects and methods for OmniDriver are organized, and then run a sample program.

Installing .NET Framework

Before you can install the OmniDriver software, you must ensure that the .NET framework has been installed. OmniDriver requires version 2.0 or later. You can download the installer for .NET 4.0 from OmniDriver CD or the Ocean Optics Software Downloads page.

If you attempt to install OmniDriver before .NET has been installed, you will see the error message: "Error running RegAsm64.exe … : Program ended with an error exit code."

Do not plug your spectrometer in until after you have installed the software.

► *Procedure*

1. Run the installer and follow the on-screen prompts.
2. Select which product you wish to install: OmniDriverSPAM, OmniDriver, or SPAM. You may only check one product.
3. Enter the usual password for installing your OmniDriver product.
4. After successfully completing the installation wizard, plug in your spectrometer and follow the procedure for installing the necessary driver for your spectrometer.

Downloading from the Ocean Optics Website

Installing on a Windows Platform

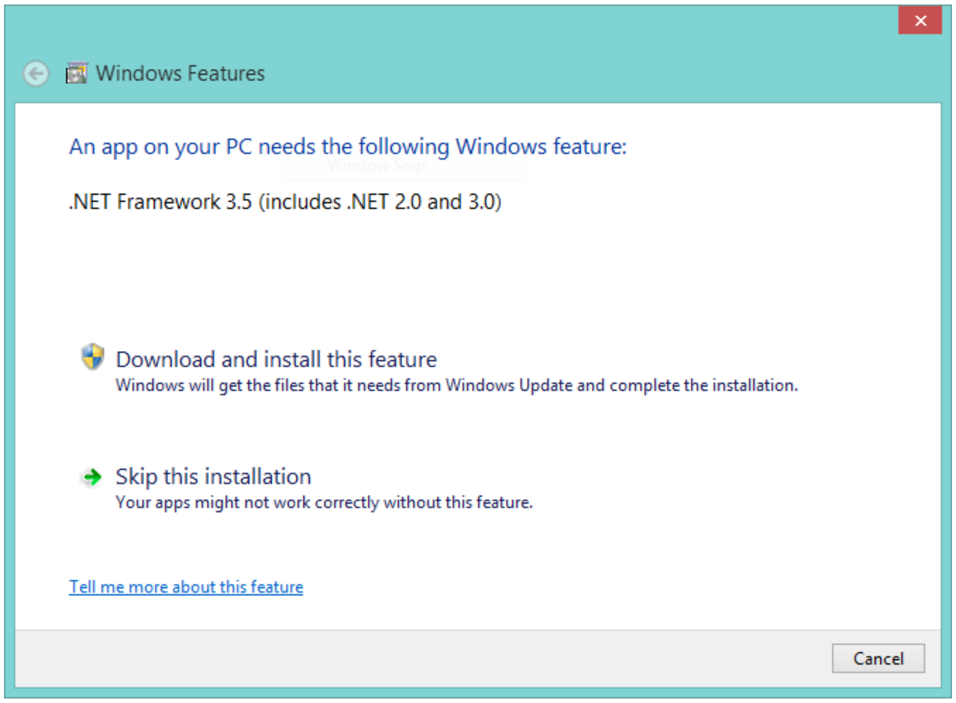
Simply download this file and double-click on it in Windows Explorer to begin the installation procedure. The installer will guide you through the install process. See *Installing on a Windows 8.1 32-bit Platform* for more information about that system.

► ***Procedure***

1. Close all other applications running on the computer.
2. Start Internet Explorer.
3. Navigate to **http://www.oceanoptics.com/technical/softwaredownloads.asp** and click on the OmniDriver software appropriate for your Windows operating system.
4. Save the software to the desired location. The default installation directory is **\Program Files\Ocean Optics\OmniDriver**.
5. The installer wizard guides you through the installation process. The OmniDriver icon location is **Start | Programs | Ocean Optics | OmniDriver | OmniDriver** and the current user's desktop.
6. Browse to **http://www.oceanoptics.com/technical/softwaredownloads.asp**.
7. Select the OmniDriver software appropriate for your operating system. The installer wizard guides you through the installation process.
8. After you have finished installing OmniDriver, you may need to install Java 1.6 JDK on your PC. (only necessary when developing in C/C++ – see *Installing the Spectrometer Driver Software*.)

Installing on a Windows 8.1 32-bit Platform

If your operating system is Windows 8.1 32-bit then you will want to run OmniDriver-x.xx-win32-installer.exe. During the installation you may be prompted with a message shown below. If so, use the following procedure.


Windows Features

An app on your PC needs the following Windows feature:

.NET Framework 3.5 (includes .NET 2.0 and 3.0)

Download and install this feature
Windows will get the files that it needs from Windows Update and complete the installation.

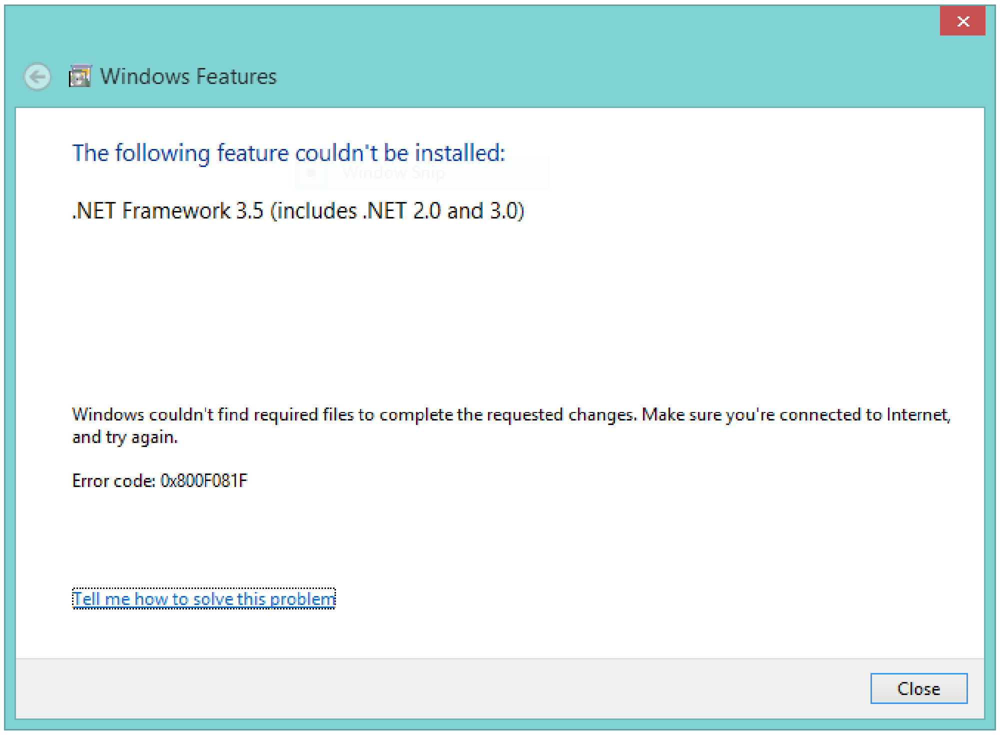
Skip this installation
Your apps might not work correctly without this feature.

Tell me more about this feature

Cancel

► ***Procedure***

1. Click **Download and install feature**.
2. If the feature fails to install, proceed anyway by clicking **Close** in the window shown below. The OmniDriver installation will still continue to completion regardless of this update.


Windows Features

The following feature couldn't be installed:

.NET Framework 3.5 (includes .NET 2.0 and 3.0)

Windows couldn't find required files to complete the requested changes. Make sure you're connected to Internet, and try again.

Error code: 0x800F081F

Tell me how to solve this problem

Close

Note

If this feature could not be installed, it is important to note that referencing NETOmniDriver-NET40.dll is needed for any .NET development (for example, using Visual Studio 2010, start a new C# Console project).

Under the reference section, add a local reference using Browse. Navigate to c:\program files\ocean optics\omnidriver\OOI_HOME, and then select **NetOmniDriver-NET40.dll**.

Installing on a Mac Platform

OmniDriver for Mac OSX is distributed as an "Application bundle", compressed into a *.tar.gz file.

► *Procedure*

The basic procedure for installing OmniDriver on the Mac is as follows:

1. Download the **OmniDriver-*n.nn*-macosx32-development-installer.app.tar.gz** file to a folder on your Mac. (*n.nn* is the version number of OmniDriver).
2. Untar the file using a command like the following:
   tar xvf OmniDriver-2.11-macosx32-development-installer.app.tar.gz
   This will create a new "folder" in the current directory containing the application bundle.
3. Using Finder, double-click on the new application bundle "folder". This will invoke the OmniDriver installer.
   You will be prompted for a password (supplied by your Ocean Optics sales/distributor).
4. The installer will perform the following operations:

   a. OmniDriver will be installed to /Applications/OmniDriver-n.nn (you may override this location).

   b. The "OOI_HOME" environment variable will be automatically defined. This environment variable points to the OOI_HOME folder where you installed OmniDriver. Typically it appears as "/Applications/OmniDriver-2.11/OOI_HOME".

   c. The OOI_HOME folder under the OmniDriver installation folder will be added to your system PATH environment variable.

   d. Symbolic links to required *.dylib files are created. Depending on the flavor of OmniDriver you are installing, you will see two or three of the following symlinks:

      i. /usr/lib/libcommon.dylib →
         /Applications/OmniDriver-*n.nn*/OOI_HOME/libcommon.dylib
      ii. /usr/lib/libOmniDriver.dylib →
         /Applications/OmniDriver-*n.nn*/OOI_HOME/libOmniDriver.dylib
      iii. /usr/lib/libSPAM.dylib →
         /Applications/OmniDriver-*n.nn*/OOI_HOME/libSPAM.dylib

5. To avoid confusion, it is a good idea to reboot your system after completing the installation. This will ensure that changes to environment variables have taken effect.

6. Verify successful installation by compiling and running the C++ sample application, located in /Applications/OmniDriver-*n.nn*/samples/cpp/GPP_SpectrumTest_Mac.

There is no need to install Java on the Mac because the required version of Java is already installed as an integral part of Mac OS X (versions 10.4 Tiger, and later). If you are running an earlier version of Mac OS X, you will need to upgrade your Java installation to at least Java 1.5.

Installing on a Linux Platform

We provide a convenient installer for OmniDriver on the Linux family of operating systems. For Linux systems, download the **OmniDriver-*n.nn*-linux*MM*-development-installer.bin** file (where *x.xx* is the version number and MM is either 32 or 64, depending on the architecture of your Linux hardware). We also provide redistributable installers which contain the minimal OmniDriver files needed to run your application on an end-user PC. The redistributable installers may be freely distributed, and they do not require a password. You may need to chmod the file to make it executable. Then run it and follow the prompts.

Note

When using OmniDriver on x64 Linux systems, you ***must*** use version 1.6.0_01 (1.6 update 1) of the Java JVM. Failure to do so will result in a segmentation fault when you attempt to run your application. This version of the JVM is now bundled with the OmniDriver installer on Linux x64 systems. Ensure that the following environment variables are defined when you build and/or run your application:

JAVA_HOME=${OMNIDRIVER_HOME}/_jvm

JDK_INCLUDE_FILE_ROOT=$HOME/jdk1.6.0_01 (or wherever you installed the Java JDK)

JVM_ROOT=${OMNIDRIVER_HOME}/_jvm

LD_LIBRARY_PATH=$OOI_HOME:$OMNIDRIVER_HOME/_jvm/lib/amd64/server

The installer will automatically define the OOI_HOME and OMNIDRIVER_HOME environment variables for you.

► *Procedure*

After the installer has finished, perform the following steps to verify that the installation was successful:

1. Plug in an Ocean Optics USB spectrometer into the USB port of your Linux-based computer.
2. Open a terminal window.
3. Run the use the lsusb command to verify that Linux sees your device. Your spectrometer will have an ID that begins with 2457. This is the Ocean Optics vendor ID.

If the computer did *not* detect your spectrometer, you must perform the following additional steps. These steps will install a rules file to tell Linux which driver to use to communicate with Ocean Optics spectrometers.

► *Procedure*

To install a rules file:

1. Open a terminal window.
2. Copy the file 10-oceanoptics.rules from the drivers directory of the OmniDriver installation area to /etc/rules.d (or /etc/udev/rules.d on Ubuntu x64).
3. Restart udev.
   On Fedora and similar Linux distros, you can restart udev with the following sequence of commands:
   a. udevcontrol reload_rules
   b. udevstart (on Ubuntu: restart udev or service udev restart)
4. If the previous command does *not* successfully restart udev, you will need to reboot your system.

After you have finished installing OmniDriver, you may need to install Java 1.6 JDK on your PC. This is only necessary if you are developing in C/C++.

Troubleshooting Your Linux Installation

On some newer distributions, problems occur when SeLinux is set to "enforcing". If you get an error containing the message: **Cannot restore segment prot after reloc: Permission denied**, you should issue this command: **hcon –t textrel_shlib_t $OMNIDRIVER_HOME/_jvm/lib/i386/client/libjvm.so**. This change only impacts OmniDriver applications.

Alternatively, to permanently change the SeLinux setting for all applications, edit the **/etc/sysconfig/selinux** file as ”root”, and change the line that says:
:

SELINUX=enforcing

to:

SELINUX=permissive

or:

SELINUX=disabled

You must reboot your computer after making this change.

Retrieving from a DVD

Your OmniDriver software may be shipped to you from Ocean Optics on a separate DVD labeled **OmniDriver/SPAM Spectroscopy Development Platform**. You will need the password located on the jacket of the CD containing your OmniDriver software to complete the installation.

Note

The redistributable files and Linux legacy files are only available from the Ocean Optics ftp site. The DVD contains the full release files.

► ***Procedure***

1. Insert the DVD that you received containing your OmniDriver software into your computer.
2. Select the OmniDriver software for your computer's operating platform. Then follow the prompts in the installation wizard.

   Or,

   Browse to the appropriate OmniDriver set-up file for your computer and double-click it to start the software installation. Set-up files are as follows:

   - Windows: OmniDriver-1.*x*-win32-installer.exe (for 32-bit platforms)
   - Windows: OmniDriver-1.*x*-win64-installer.exe (for 64-bit platforms)
   - Mac: OmniDriver_1.*x*_Mac_full_release.dmg
   - Linux: OmniDriver-1.*x*-linux-installer.bin (for 32-bit platforms)
   - Linux: OmniDriver-1.*x*-linux64-installer.bin (for 64-bit platforms)
3. Refer to the appropriate installation section, depending on your operating platform to complete the installation. See *Installing on a Windows Platform*, ***Error! Reference source not found.***, *Installing on a Mac Platform*, or *Installing on a Linux Platform* and follow the steps listed in that section.

Installing the Spectrometer Driver Software

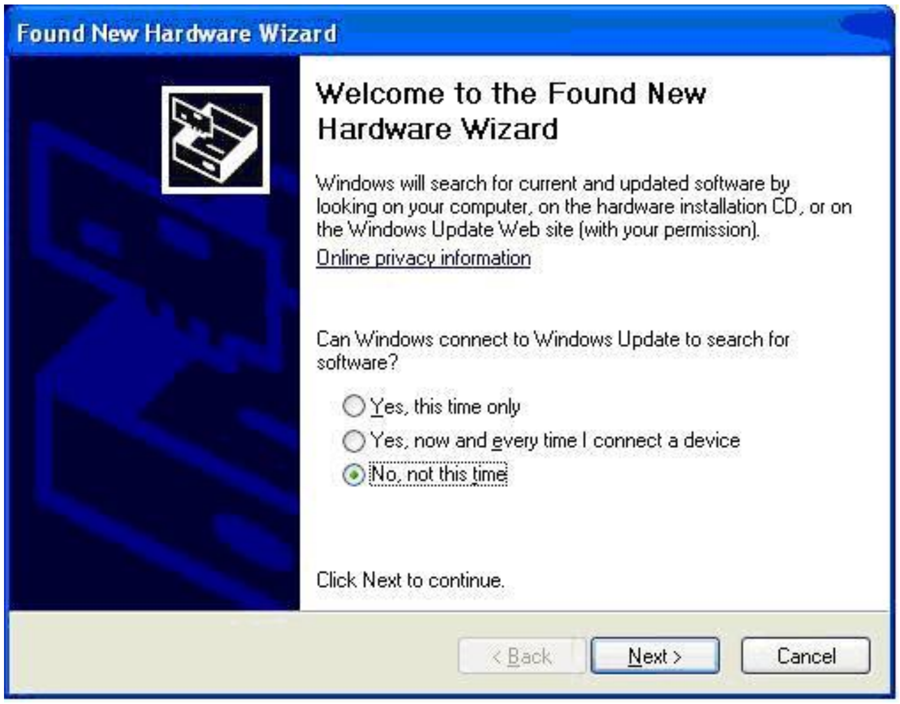
Do not plug your spectrometer in until after you have finished installing the OmniDriver software on your system. Installing the spectrometer driver software is straightforward for 32-bit systems. Follow the wizard prompts. The following instructions help guide you through installing the spectrometer driver for 64-bit systems. For 64-bit systems, the procedure differs depending on whether you are using Windows XP (64-bit) or Windows 7 (64-bit).

For QE *Pro* spectrometers, see *QE Pro Spectrometer Device Driver Installation*.

Installing Driver Software on Windows XP (64-bit)

► ***Procedure***

1. When you first plug in your spectrometer, Windows XP displays the following screen. Choose **No, not this time**.


Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).

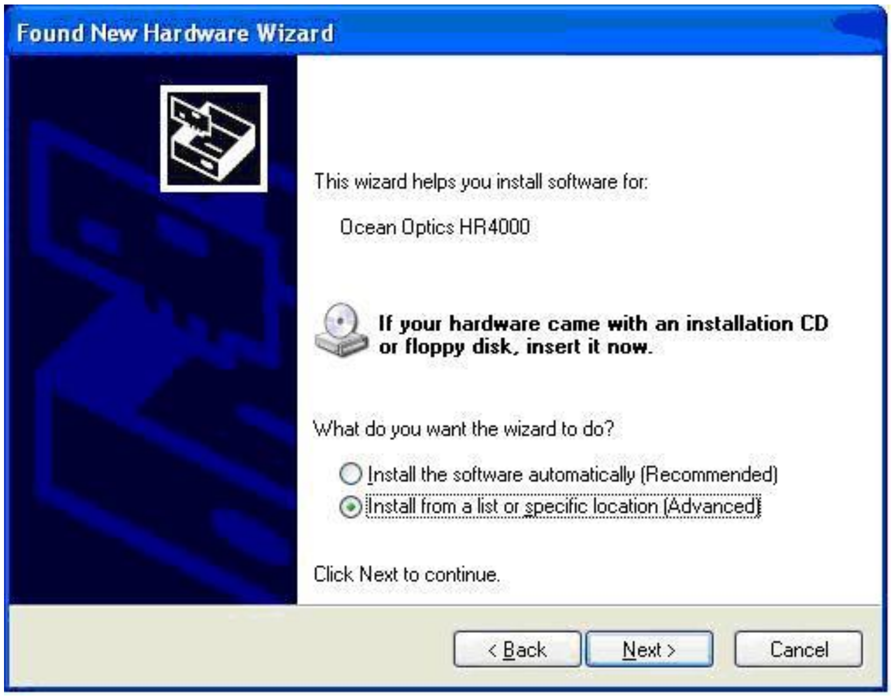
Online privacy information

Can Windows connect to Windows Update to search for software?

Yes, this time only
Yes, now and every time I connect a device
[✓ No, not this time]

Click Next to continue.

2. Choose **Install from a list or specific location (Advanced)**.


Found New Hardware Wizard

This wizard helps you install software for:

Ocean Optics HR4000

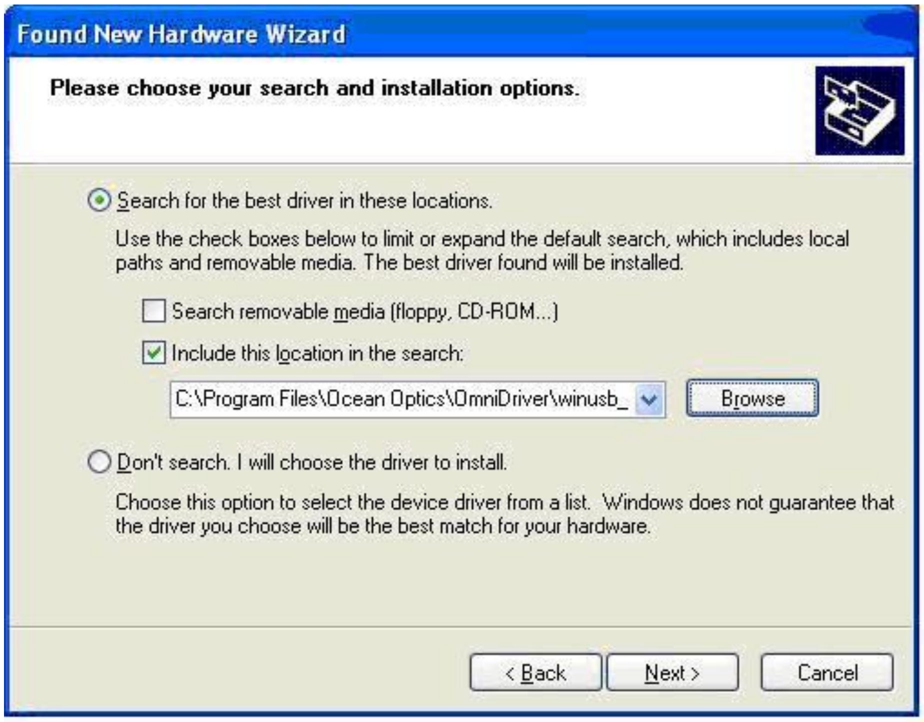
If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

Install the software automatically (Recommended)
[✓ Install from a list or specific location (Advanced)]

Click Next to continue.

3. Navigate to C:\Program Files\Ocean Optics\OmniDriverSPAM\winusb_drivers.



Found New Hardware Wizard

Please choose your search and installation options.

(●) Search for the best driver in these locations.

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

☐ Search removable media (floppy, CD-ROM...)

☑ Include this location in the search:

C:\Program Files\Ocean Optics\OmniDriver\winusb_

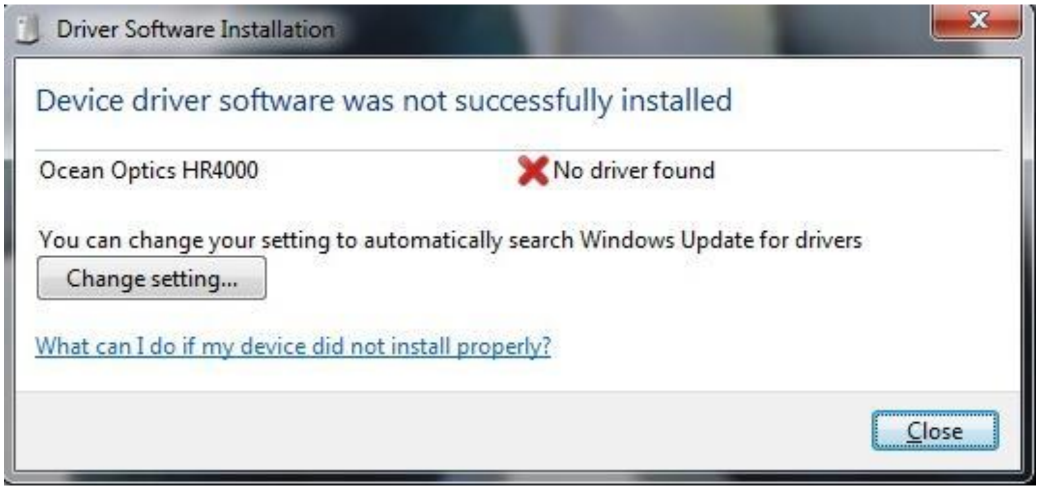
( ) Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

4. The Progress window appears. When the installation has completed, the final window appears. Click **Finish** to complete the installation.

Installing the Driver Software on Windows 7 (64-bit)

When you first plug in your spectrometer, Windows 7 will attempt (unsuccessfully) to automatically install the driver for your spectrometer. Click close to close the warning message:



Driver Software Installation

Device driver software was not successfully installed

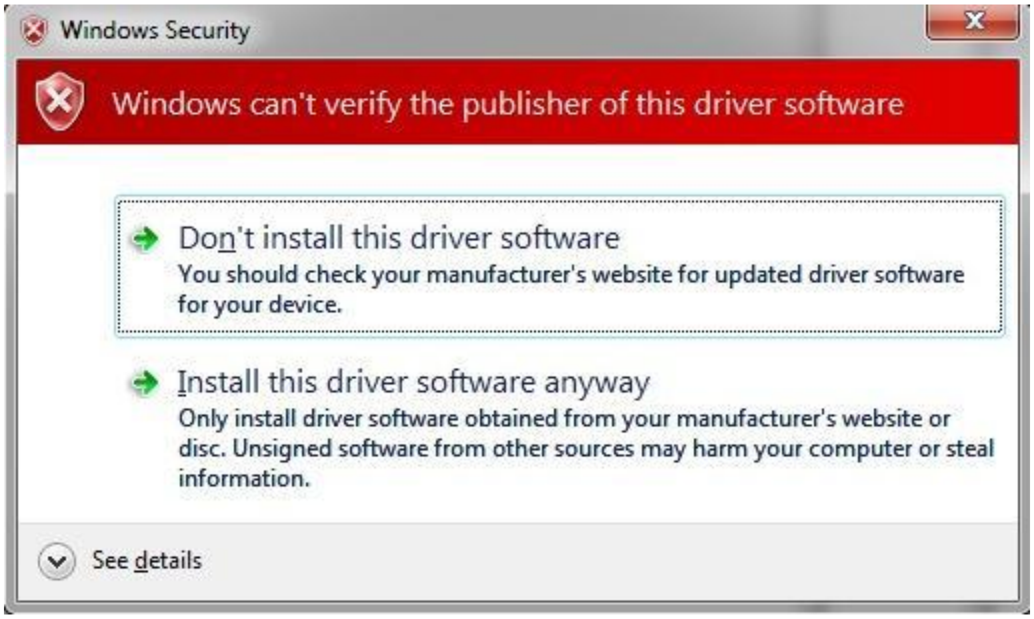
Ocean Optics HR4000	✗ No driver found

You can change your setting to automatically search Windows Update for drivers

► ***Procedure***

To manually install the new 64-bit driver for your spectrometer:

1. Make sure your spectrometer is plugged in.
2. Select Start | Control Panel | Device Manager.
3. Right-click on your spectrometer and choose **Update Driver Software…**.
4. Choose the Browse my computer for driver software option.
5. Highlight the **winusb_driver** directory and click **OK**.
   (located at C:\Program Files\Ocean Optics\OmniDriverSPAM\winusb_driver)
6. Click **Next**. On Windows 7 systems, you will see the following Windows Security warning:



Windows Security

Windows can't verify the publisher of this driver software

Don't install this driver software
You should check your manufacturer's website for updated driver software for your device.

Install this driver software anyway
Only install driver software obtained from your manufacturer's website or disc. Unsigned software from other sources may harm your computer or steal information.

See details

7. Choose Install this driver software anyway.
8. Verify your installation was successful by running the SpectrumTest64.exe application (located in the C:\Program Files\Ocean Optics\OmniDriverSPAM\OOI_HOME directory).

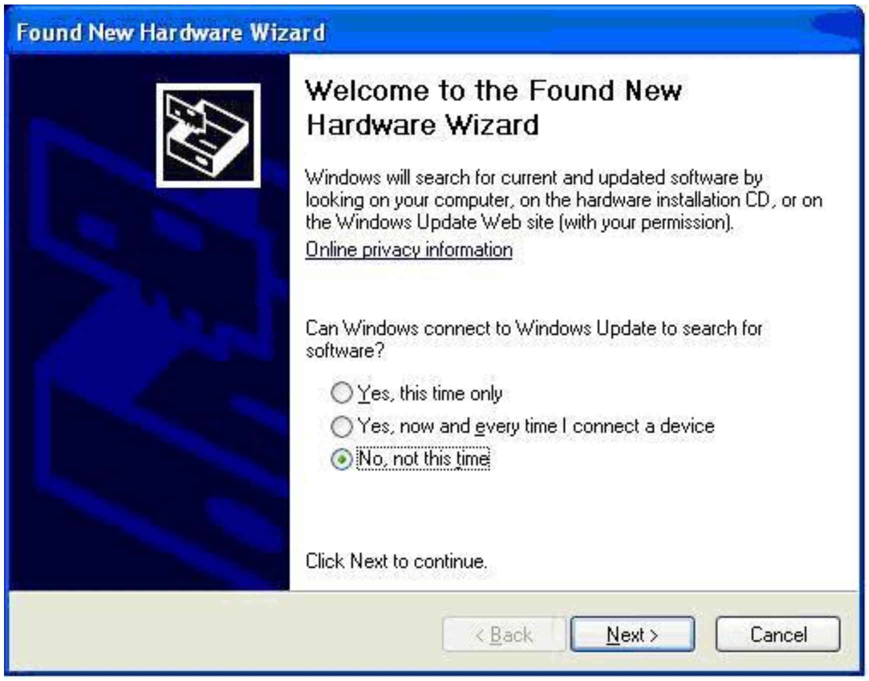
QE *Pro* Spectrometer Device Driver Installation

The QE Pro spectrometer requires special device driver installation instructions. Choose the appropriate instructions listed below depending on the type of Windows system your computer is running.

Installing the Driver Software for QE *Pro* Spectrometers on on Windows XP (32 and 64-bit)

► *Procedure*

1. When you first plug in your spectrometer, Windows XP displays the following screen. Choose **No, not this time**.



Welcome to the Found New Hardware Wizard

Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).

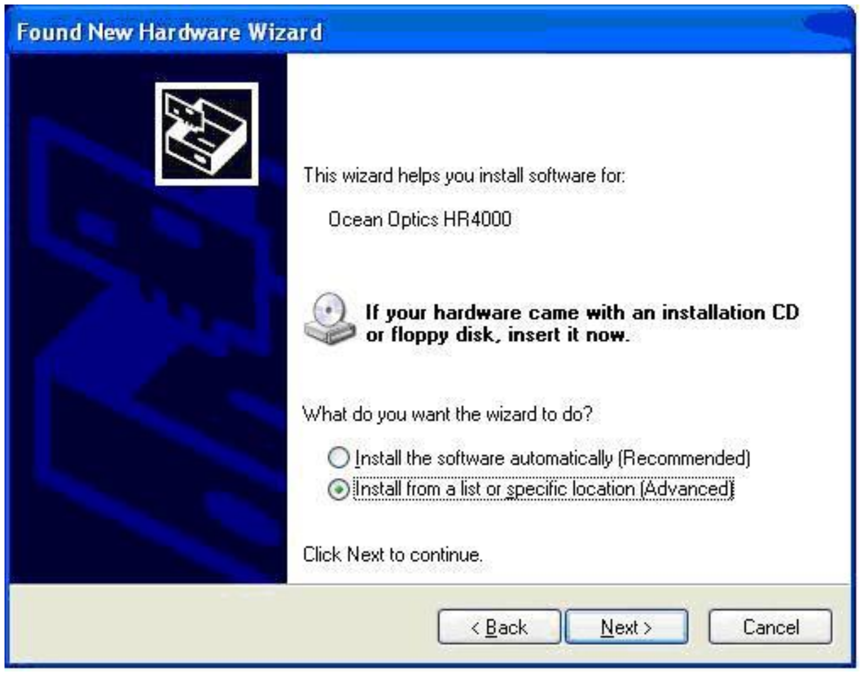
Online privacy information

Can Windows connect to Windows Update to search for software?

Yes, this time only
Yes, now and every time I connect a device
[✓ No, not this time]

Click Next to continue.

2. Choose **Install from a list or specific location (Advanced)**.



This wizard helps you install software for:

Ocean Optics HR4000

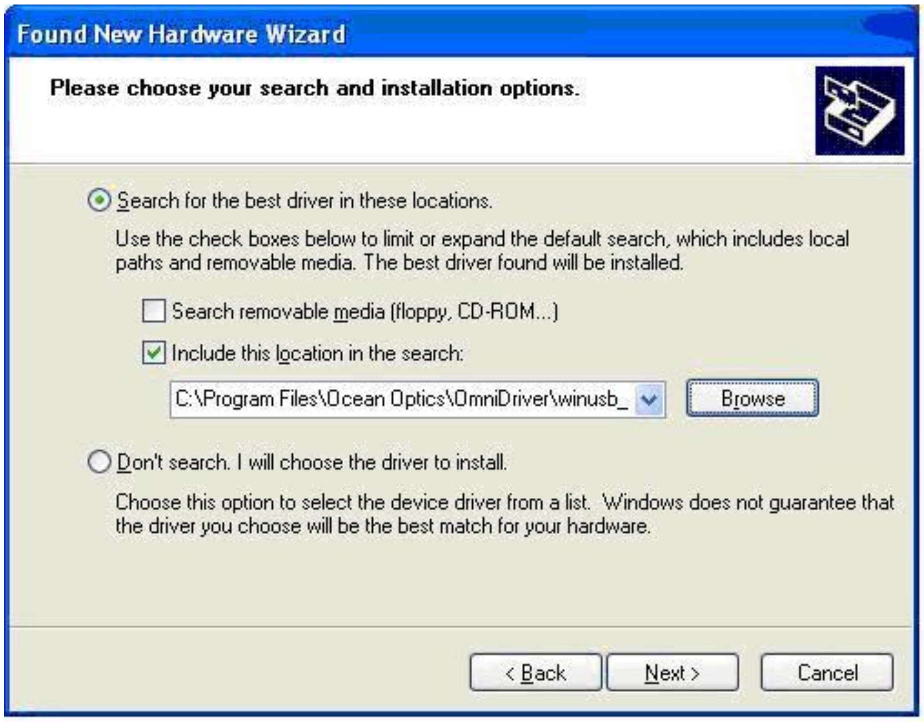
If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

Install the software automatically (Recommended)
[✓ Install from a list or specific location (Advanced)]

Click Next to continue.

3. Navigate to C:\Program Files\Ocean Optics\OmniDriver\winusb_drivers.


Found New Hardware Wizard
Please choose your search and installation options.
< Back
Next >
Cancel
Browse

(●) Search for the best driver in these locations.

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

☐ Search removable media (floppy, CD-ROM...)

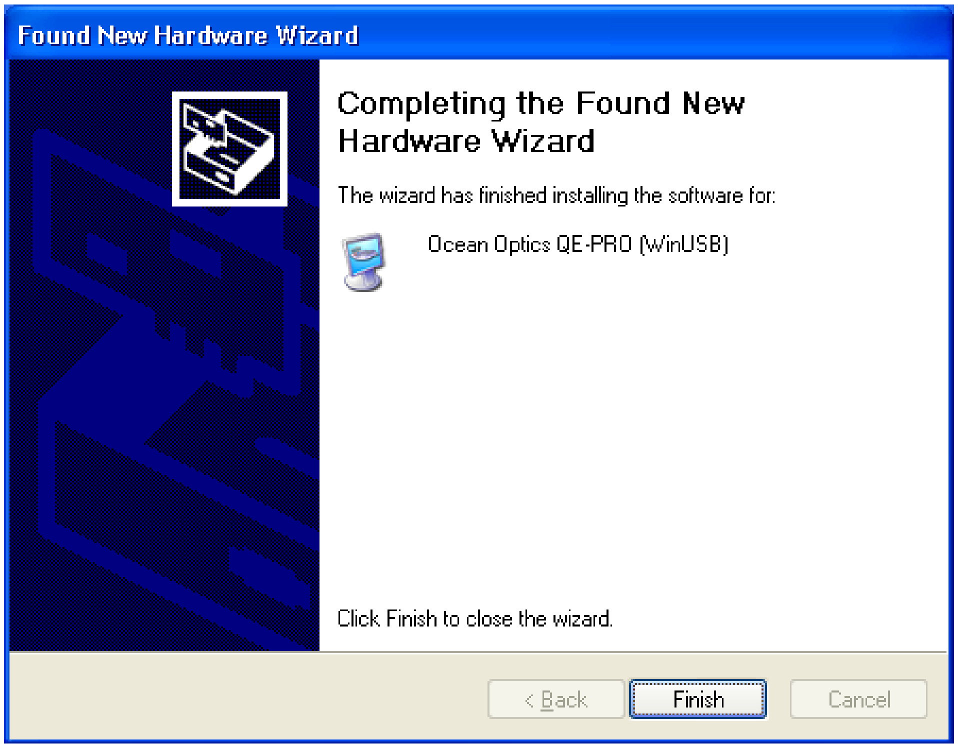
☑ Include this location in the search:

C:\Program Files\Ocean Optics\OmniDriver\winusb_

( ) Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

4. The Progress window appears. When the installation has completed, the final window appears. Click **Finish** to complete the installation.



Completing the Found New Hardware Wizard

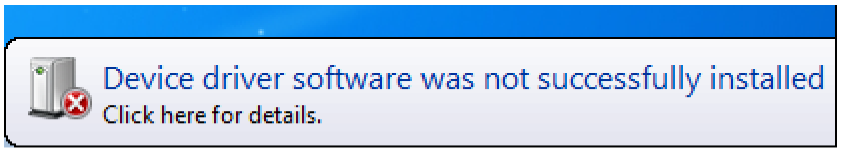
The wizard has finished installing the software for:

Ocean Optics QE-PRO (WinUSB)

Click Finish to close the wizard.

Installing the Driver Software for QE *Pro* Spectrometers on Windows 7 (32 and 64-bit)

When you first plug in your spectrometer, Windows 7 will attempt (unsuccessfully) to automatically install the driver for your spectrometer. Click close to close the warning message:



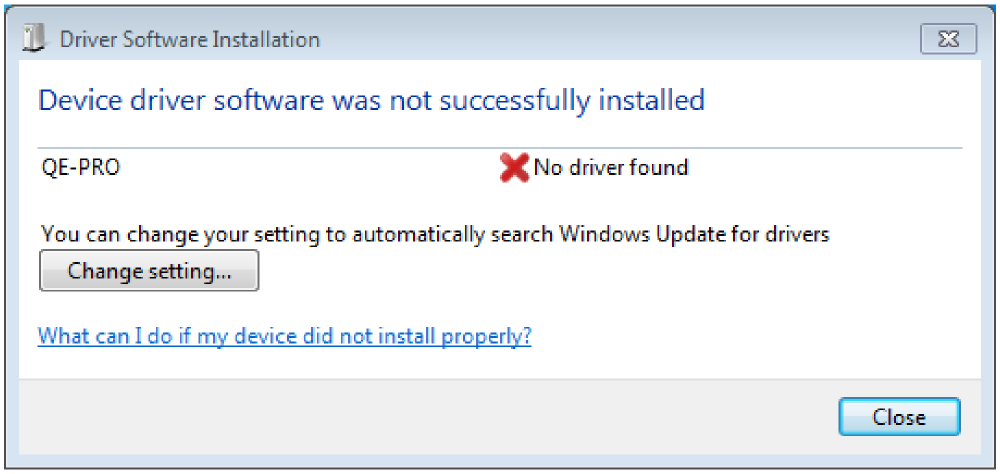
Device driver software was not successfully installed

Click here for details.

► *Procedure*

To manually install the new 64-bit driver for your spectrometer:

1. Click **Change setting**.



Driver Software Installation

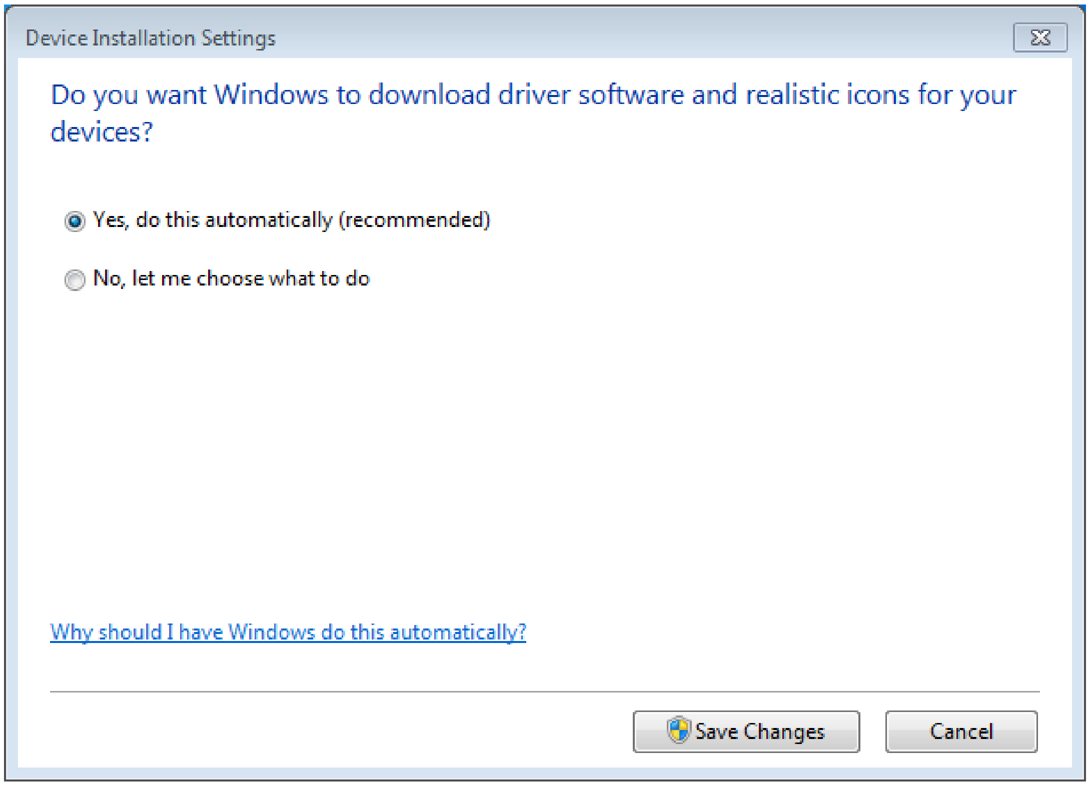
Device driver software was not successfully installed

QE-PRO ❌ No driver found

You can change your setting to automatically search Windows Update for drivers

Change setting...
What can I do if my device did not install properly?
Close

2. Select **Yes, do this automatically**.



Device Installation Settings

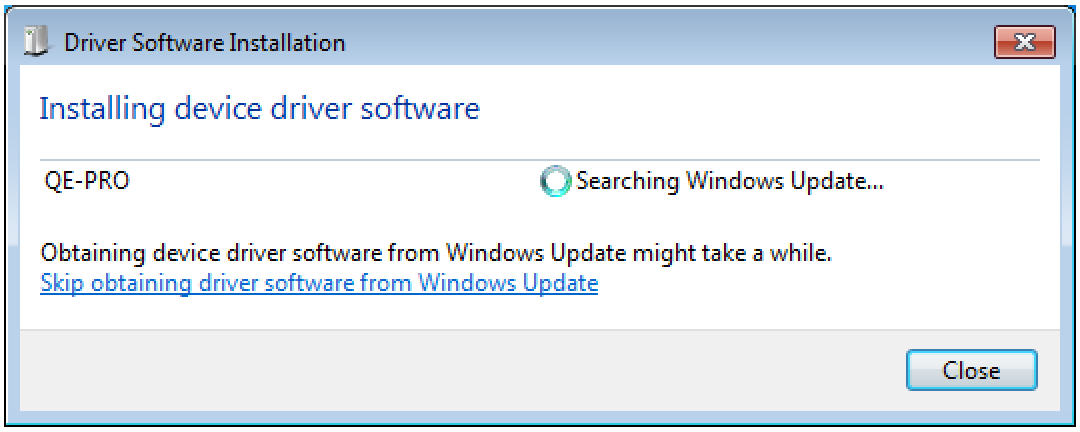
Do you want Windows to download driver software and realistic icons for your devices?

(●) Yes, do this automatically (recommended)

( ) No, let me choose what to do

Why should I have Windows do this automatically?

The system installs the device driver software.



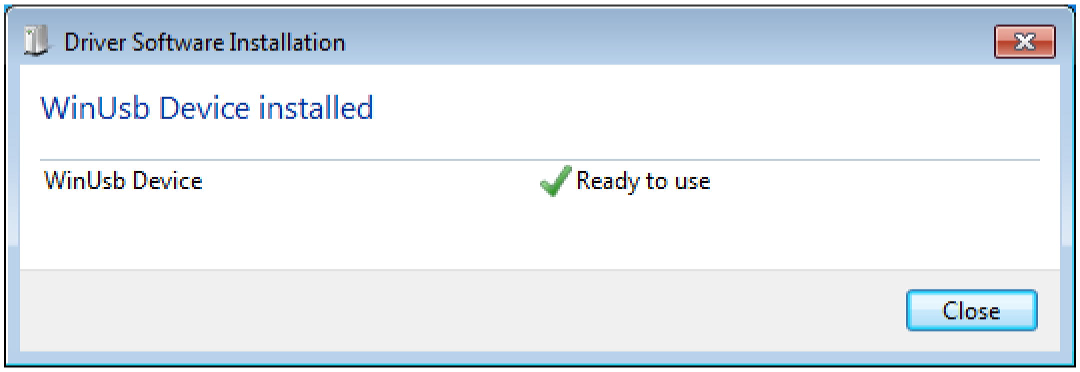
Driver Software Installation

Installing device driver software

QE-PRO Searching Windows Update...

Obtaining device driver software from Windows Update might take a while.
Skip obtaining driver software from Windows Update

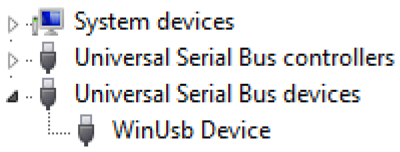
When the installation process has finished successfully, the following screen displays:


Driver Software Installation

WinUsb Device installed

WinUsb Device ✓ Ready to use

The Device Manager reports the following information:


System devices
Universal Serial Bus controllers
Universal Serial Bus devices
WinUsb Device

Uninstalling an Incorrect Driver

If you need to uninstall an incorrect driver, use the Windows Device Manager.

► *Procedure*

1. Right-click on **My Computer** and choose **Properties**.
2. Select the **Hardware** tab and then click the **Device Manager** button.
3. Expand the Ocean Optics section.
4. Right-click on the offending spectrometer and choose **Uninstall**.
5. Copy the INF/SYS files.
6. Plug the spectrometer back in and follow the steps given in the Found New Hardware wizard.

Installing Java

OmniDriver comes bundled with a copy of the Java 1.6 Run-time Engine (JRE). Therefore, in most cases, it is not necessary to manually install Java on your computer. However, if you are developing OmniDriver applications using the C/C++ languages and you are using the C/C++ interface to OmniDriver, it will be necessary for you to install the Java 1.6 JDK (Java Developer's Kit). The Java JDK contains header files required for C/C++ development. You can obtain the Java 1.6 JDK from **www.javasoft.com**. There is no charge for this software.

If you are using the COM or .NET interface to OmniDriver, you do NOT need to install the Java JDK.

Note

Ocean Optics provides a royalty-free, redistributable installer that you may use to install the run-time components of OmniDriver on your end-user's computer. This installer contains all necessary Java files. Therefore, if you use this redistributable installer, you should never need to perform a separate install of Java on your end-user's computer.

Java JDK

If you are developing applications in C/C++, or you are using the C/C++ OmniDriver interface, you will need to download and install the 64-bit version of the Java JDK. This will give you the header files needed when you compile your OmniDriver application.

OmniDriver comes bundled with its own copy of the 64-bit Java JRE. The only reason you would need to install the JDK is to access the header files provided by the JDK (required to compile your OmniDriver application).

Installation Troubleshooting Notes

Problem	Possible Cause(s)	Suggested Solution(s)
You have installed the latest version of OmniDriver while your computer is plugged in to your spectrometer, but your application does not see it.	The old driver (ezusb.sys) for your spectrometer must be uninstalled.	Uninstall the old spectrometer driver.
You have installed the latest version of OmniDriver and plugged in your spectrometer, but your application does not see it.	The USB device needs to be enumerated	Unplug the USB cable. Wait 5 seconds. Plug the USB cable back in.
UnsatisfiedLinkError message	The new WinUSB driver is not installed yet.	Install the WinUSB driver.
While installing driver, an error occurs concerning an incorrect co-installer path.	Device co-installer path is incorrect.	Change the end of path from i386 to x86.

Chapter 3

Basic Sequence of Operations

Overview

This chapter describes the typical sequence of operations that the application must perform to control a spectrometer and acquire spectra. The functions described here are common to all Ocean Optics spectrometers. Additionally, more detailed information about each operation is in Chapter 11: *Wrapper API Reference*. This chapter provides exact syntax for each method call, including all parameters and data types.

In addition to the basic capabilities provided by all Ocean Optics spectrometers, there are also a number of optional features offered by some, but not all, spectrometers. Refer to *Optional Features* for more information.

Create/Initialize the Wrapper Object

Before you can control your spectrometer, you must create an instance of the Wrapper object. This is your gateway into all of the capabilities of the spectrometer.

Caution

Your application must create only ONE instance of the wrapper object. This wrapper object is then shared by all spectrometers under the control of your application. Any and all threads created by your application must then share a reference to the same instance of the wrapper object. Furthermore, there may only be ONE application (EXE) running on your computer that creates a wrapper object and controls spectrometers. All interaction with all Ocean Optics spectrometers attached to your computer must be performed within this one single executable/application.

C++

```
Wrapper wrapper; // simply declare an instance
```

Visual Basic/COM

```
Dim wrapper as New OmniDriver.CCoWrapper
```

Open All Spectrometers

Next, you must query the USB port(s) to discover all attached spectrometers. Behind the scenes, the `openAllSpectrometers()` method attempts to communicate with a potential Ocean Optics spectrometer at every available USB address. If we receive a reply to our query, we conclude that a spectrometer is present at that USB position.

The `openAllSpectrometers()` method normally returns an integer from 0 – N, indicating the number of USB spectrometers it found. If an I/O error occurred, this method returns “-1”. In this case, you can call the wrapper.getLastException() function to learn more about the nature of the error.

Note for Jaz Network Spectrometer

The Jaz network spectrometer (Jaz connected via Ethernet) communicates over the internet rather than USB. Obviously we cannot ping every IP address in the world to discover all Jaz-network devices, so we provide the method `openNetworkSpectrometer(String ipAddress)` to access your Jaz network devices. If you only have Jaz network spectrometers and do not have any USB spectrometers, you do not need to call the `openAllSpectrometers()` method. In this case you only need to call `openNetworkSpectrometer()`. It does not matter whether you call `openNetworkSpectrometer()` before or after calling `openAllSpectrometers()`. Once you have opened the Jaz network spectrometer, you can use all of the other wrapper functions just as you would for any other type of spectrometer.

C++

```
int numberOfSpectrometers;
int spectrometerIndex;
spectrometerIndex =
   wrapper.openNetworkSpectrometer("192.168.2.1");
numberOfSpectrometers = wrapper.openAllSpectrometers();
numberOfSpectrometers =
   wrapper.getNumberOfSpectrometersFound();
```

Visual Basic - COM interface

```
Dim numberOfSpectrometers as Integer
Dim spectrometerIndex as Integer
numberOfSpectrometers = wrapper.openAllSpectrometers()
If numberOfSpectrometers = -1 Then
   MsgBox("I/O error occurred")
End If
spectrometerIndex =
   wrapper.openNetworkSpectrometer("192.168.2.1");
If spectrometerIndex = -1 Then
   MsgBox("I/O error occurred")
End If
```

Get Identification Information

This step is optional.

If you wish to identify which spectrometer you are communicating with, there are two primary methods that help identify the spectrometer:

- `wrapper.`**`getName`**`(int spectrometerIndex)` returns the type of the spectrometer (eg. "USB4000').
- `wrapper.`**`getSerialNumber`**`(int spectrometerIndex)` returns the unique serial number assigned to your spectrometer.

Set Acquisition Parameters

Normally you will want to set one or more acquisition parameters controlling the conditions under which spectrum samples are obtained. Chapter 4: *Acquisition Parameters* lists all available parameters. Here we show only the most basic acquisition parameter, wrapper `setIntegrationTime` (`int spectrometerIndex`, `int integrationTime`), and how to set it.

C++

```
int integrationTime; // units: microseconds
int spectrometerIndex; // 0-n; selects which spectrometer // you are talking
   //to
integrationTime = 5000; // 5 milliseconds
spectrometerIndex = 0;
wrapper.setIntegrationTime(spectrometerIndex, integrationTime);
```

Visual Basic - COM interface

```
Dim integrationTime as Integer ‘ units: microseconds
Dim spectrometerIndex as Integer ‘ 0-n; selects which
                              ‘ spectrometer you are talking to
integrationTime = 5000 ‘ 5 milliseconds
spectrometerIndex = 0
wrapper.setIntegrationTime(spectrometerIndex, integrationTime)
```

Acquire a Spectrum

Now you are ready to acquire a spectrum. A spectrum is simply a one-dimensional array of pixel values, stored as "doubles". The number of elements in this array vary for each spectrometer. Dark pixels, if provided by your spectrometer, are also returned within this array. See the documentation for your spectrometer to identify the location of the dark and active pixels within this array.

The function call is `wrapper.`**`getSpectrum`**`(int spectrometerIndex).`

When you call the `getSpectrum()` method, the spectrometer will return the next available spectrum to your application, assuming the spectrometer is in its default normal mode. In this mode, the spectrometer is continuously acquiring new spectra. Thus, if you happen to call `getSpectrum()` when the spectrometer has nearly completed the acquisition of a spectrum, it may very well return almost immediately, even if you have specified a long integration period. This is normal behavior and is not indicative of a problem.

Additionally, the duration of your call to `getSpectrum()` could be delayed if it is necessary for OmniDriver to take a stability scan. A stability scan is required whenever you change an acquisition parameter that could affect the data values of the spectrum, such as the integration time. Because the spectrometer is always acquiring spectra, it must wait until the current spectral acquisition (which is still using the old integration time) has completed. This spectrum will be "thrown away". Then the spectrometer must begin acquiring a second spectrum using your new integration time setting. So potentially, two complete spectra may have been acquired before `getSpectrum()` returns to the caller.

See Appendix A, "FAQ: Why doesn't getSpectrum() return when I think it should" for a more complete discussion of the timing of the getSpectrum() method.

C++

```
#include "ArrayTypes.h"
int numberOfPixels;
DoubleArray pixelArray;
int spectrometerIndex; // 0-n
pixelArray = wrapper.getSpectrum(spectrometerIndex)
numberOfPixels = pixelArray.getLength();
// Convert the DoubleArray object into an array of
// double primitives
double* pixelValues = pixelArray.getDoubleValues();
printf("pixel[0] = %f\n",pixelValues[0]);
```

Visual Basic - COM interface

```
Dim pixels() as Double
Dim spectrometerIndex as Integer ' 0-n
pixels = wrapper.getSpectrum(spectrometerIndex)
Console.WriteLine("pixel(0) = " & pixels(0))
```

Close All Spectrometers

When your application is ready to terminate, call the `wrapper.closeAllSpectrometers()` method.

Chapter 4

Acquisition Parameters

Overview

This chapter presents a complete list of all available acquisition parameters. An acquisition parameter is a setting that controls some aspect of the conditions under which a spectrum is acquired. All spectrometers support these acquisition parameters.

Auto Toggle Strobe Lamp Enable

When auto-toggling is enabled (*and* `setStrobeEnable()` is set to true), the lamp will be automatically turned on for the duration of each spectrum acquisition, and then turned off until the next spectrum is acquired. Use auto-toggling to avoid or delay deterioration of specimens that are sensitive to light. It can also extend the life of your lamp and prevent premature deviation from the specification sheet for calibrated light sources.

Note

You must also call `wrapper.setStrobeEnable(true)` or this function will have no effect.

Function call:

```
wrapper.setAutoToggleStrobeLampEnable(spectrometerIndex,enableFlag)
```

Boxcar Width

Boxcar smoothing “smoothes” the spectrum pixel values (i.e., reduces noise) by averaging a range of contiguous pixels together. Each pixel in the spectrum is averaged with *n* pixels on either side of it. A new spectrum is generated containing these averaged pixel values.

Function call:

```
wrapper.setBoxcarWidth(spectrometerIndex,numberOfPixelsOnEitherSideOfCenter)
```

Correct for Detector Nonlinearity

All Ocean Optics spectrometers are calibrated at the factory to maximize accuracy. One of the calibrations performed is to correct for detector nonlinearity.

This calibration consists of eight numbers used as the coefficients of a 7th order polynomial that adjusts for the fact that the CCD detectors don't respond to stimuli photons uniformly as more and more electrons are drained from the well. In other words, the efficiency of the CCD detectors may be 30% when the well is half full, but may be only 20% when the well is completely drained of electrons.

By "efficiency" we mean the probability that an incoming photon will drain an electron from the CCD well; 100% efficiency means every incoming photon will drain one electron, while 50% efficiency means an incoming photon has only a 50% chance of causing an electron to drain.

Nonlinearity calibration is made by averaging together all pixels of the CCD array. Thus, we are assuming that all pixels respond about the same.

Function call:

```
wrapper.setCorrectForDetectorNonlinearity(spectrometerIndex,enableFlag)
```

Correct for Electrical Dark

Many (not all) spectrometers have a number of dark pixels. A dark pixel is an actual electrically active CCD detector pixel which has been physically coated to prevent any light from entering. Due to leakage, a small amount of electrons escape from the well even if there is absolutely no light. Electric dark correction computes an average value of these dark pixels over fifteen consecutive scans, and then subtracts this average value from all pixels in the spectrum to eliminate error due to leakage. Although the algorithm specifies that 15 consecutive scans be averaged together, the correction factor will be applied as soon as the first spectrum of data becomes available. Then additional scans will be averaged into the correction factor as these spectra become available. A running average of the most recent 15 scans is maintained.

Since the amount of leakage is proportional to the duration of the integration time, whenever the integration time is changed, the previous correction factor is discarded and we begin to compute a new correction factor based entirely on the new integration time.

Function call:

```
wrapper.setCorrectForElectricalDark(spectrometerIndex,enableFlag)
```

Integration Time

Integration time is simply the length of time during which we allow light to pass into the spectrometer's detector. In low-light level situations you may want to lengthen this period to obtain a meaningful spectrum. In high-light level situations you may want to shorten this period to avoid saturation. Saturation occurs when the one or more CCD pixels or wells have been completely drained (or completely filled on some detector models).When this occurs, additional photons entering the "well" have no effect, and the spectrum becomes increasingly distorted. If you plot a saturated spectrum on a graph, it will appear clipped at the peaks.

Integration time is specified in units of microseconds. See the documentation for your spectrometer to determine the allowable minimum and maximum integration times it will support.

Function call:

```
wrapper.setIntegrationTime(spectrometerIndex,integrationTimeInMicrosec
onds)
```

Scans to Average

Scans to average is another method used to perform noise-reduction (smoothing) on the spectra returned by a spectrometer. With this technique, multiple sequential spectra are averaged together to produce a single averaged spectrum. The algorithm uses corresponding pixels from each spectrum when computing the average for a given pixel value. For example, if Scans to Average is set to 5, the pixel[0] values from each of five consecutive scans are added together, and then divided by 5. The resulting value will be reported in pixel[0] of the spectrum returned to the user. This procedure is repeated for each pixel in the spectrum.

If you have specified a large number of scans to average, or you are using a relatively long integration time, it may take a considerable length of time to compute the smoothed spectrum. If during this time you decide to abort the operation, you do not need to wait for all of the contributory spectra to be acquired. Calling `wrapper.stopAveraging()` causes `wrapper.getSpectrum()` to return immediately (as soon as the current raw spectrum has been acquired). Of course, your application must be multi-threaded to take advantage of this ability to abort averaging since the original call to `getSpectrum()` will block the thread from which it was called.

Note

In the following function call, if you specify a value of **1** for the `numberOfScansToAverageTogether` argument, no smoothing will be performed. Each spectrum will be reported as is, without any averaging.

Function call:

```
wrapper.setScansToAverage(spectrometerIndex,numberOfScansToAverageTogether)
```

Strobe Enable

This parameter controls whether a light source is turned on or off. Ocean Optics offers a number of light sources that can be controlled by your OmniDriver application. These light sources attach directly to an electrical connector on your spectrometer. You can then call the `setStrobeEnable()` function to turn the light source on and off.

Note

The lamp does *not* respond immediately when you call this function. Rather, the lamp will change state when you call `wrapper.getSpectrum()` to request a new spectrum. So if you call `setStrobeEnable()`, but fail to call `getSpectrum()`, the lamp will *never* change state to match the state you requested.

Function call:

```
wrapper.setStrobeEnable(spectrometerIndex,enableFlag)
```

Chapter 5

Optional Features

Overview

A feature is defined as an optional capability offered by some spectrometers, but not available on all spectrometers. All spectrometers support certain basic operations, such as setting the integration time and acquiring a spectrum. But not all spectrometers support all features. See your spectrometer's documentation to determine which features it supports.

Another characteristic of a feature is that the methods or functions you must call to use it are not provided directly in the Wrapper class. Instead, for each feature, Wrapper provides a pair of related methods you can call to determine if a feature is supported by your spectrometer and to obtain a secondary object with the methods needed to use or control that feature.

Thus, in the Wrapper object, you will see a number of methods whose names look something like `isFeatureSupportedXXX()` where "*XXX*" is the name of the feature. These methods return a true/false value to indicate whether that spectrometer supports that feature. It is very important to always call the `isFeatureSupportedXXX()` method *before* attempting to use a feature. If you attempt to use a feature that is not supported by your spectrometer, nothing will happen.

After calling the `isFeatureSupportedXXX()` method (and assuming it returns a "true" value), the second step is to obtain a secondary object containing the methods unique to that feature. Again, in the Wrapper object, you will see a number of methods whose names look something like `getFeatureControllerXXX()` where *XXX* is the name of the feature. Each of these methods returns a different object, unique to that feature. For example, `getFeatureControllerThermoElectric()` returns the ThermoElectric object. The ThermoElectric object has methods that allow you to set the desired detector temperature in your spectrometer.

Caution

Never create instances of a feature-controller object directly. Such an object will not be connected to a spectrometer and calling its methods will have no effect. Always use the wrapper `getFeatureControllerXXX()` methods to obtain instances of feature-controller objects.

The following sections describe all of the optional features, but be sure to see the documentation for your spectrometer to determine if it supports that specific feature.

Analog In

For information about this feature, see the documentation for your spectrometer. Additionally, the OmniDriver javadocs for the AnalogIn class provide details on how to use this feature.

Analog Out

For information about this feature, see the documentation for your spectrometer. Additionally, the OmniDriver javadocs for the AnalogOut class provide details on how to use this feature.

Board Temperature

Some spectrometers contain a temperature sensor chip mounted on the printed circuit board (PCB). Your application can read out this temperature value, in degrees Celsius, by means of this feature.

Do not confuse this feature with the *Thermo Electric Cooling* feature, described later in this chapter.

Java

```
BoardTemperature boardTemperature;
int spectrometerIndex;
double temperatureCelsius;
spectrometerIndex = 0; // set this to the index of your spectrometer
if (wrapper.isFeatureSupportedBoardTemperature(spectrometerIndex) == true)
   {
      // Board temperature feature is supported by this spectrometer
      boardTemperature =
       wrapper.getFeatureControllerBoardTemperature(spectrometerIndex);
      try {
         temperatureCelsius =
            boardTemperature.getBoardTemperatureCelsius();
         System.out.println("board temperature = " +
            temperatureCelsius);
      } catch (IOException ioException) {
         System.out.println("Exception occurred);
         System.out.println(ioException);
`  }
} else {
      System.out.println("Board temperature feature is not supported by
         this spectrometer.");
}
```

Continuous Strobe

Continuous Strobe feature gives you the ability to generate a continuous square-wave with a 50% duty cycle. This square wave is made available to an attached Ocean Optics light source, or to your own custom hardware, via one of the pins on the spectrometer's external connector. The signal is disabled until the start of the integration period. When the integration period begins, the continuous-strobe signal is enabled and you will see it on the appropriate connector pin of your spectrometer.

The phase of the continuous-strobe signal is not synchronized with the start of the integration period. In other words, when the integration period begins, the continuous strobe signal may be anywhere within its cycle, and the voltage level on the output pin at that moment may be low or high.

To set the duration of the *full* period of the continuous-strobe signal, call `continuousStrobe.`**`setContinuousStrobeDelay`**`(int durationOfPeriodInMicroseconds)`. The allowed range of values is 0 to 65535. Thus, if you want the signal to be high for 10 milliseconds and then low for 10 milliseconds, you should specify the value 20000 (20 milliseconds, converted to microseconds).

Note

You will not see a signal on the continuous-strobe pin until you have called `wrapper.setStrobeEnable(true)`. You may call `wrapper.setStrobeEnable(true)` before or after setting the Continuous Strobe Delay parameter; it makes no difference to the software. However, it is cleaner if you leave the lamp turned off (`wrapper.setStrobeEnable(false)`) until you have set the value of your Continuous Strobe Delay parameter.

Caution

- **Not all spectrometers support the continuous strobe signal. Check your spectrometer's documentation to verify whether it is supported.**
- **Not all lamps support Continuous Strobe mode. Check your spectrometer's documentation to verify the availability of this feature.**
- **Some lamps, such as the Ocean Optics PX2 light source are strobe lamps. With these lamps, the duty cycle will not be 50%. Rather, the duration of the light pulse will be fairly instantaneous, defined by the characteristics of the strobe lamp itself.**

Visual Basic

```
Dim continuousStrobeFeature As OmniDriver.CCoContinuousStrobe
Dim period As Integer
Dim spectrometerIndex As Integer
spectrometerIndex = 0 ' arbitrarily choose the first spectrometer
' The following constant defines the period of one complete
' on/off cycle of the continuous-strobe signal
' The duty cycle is always 50% (ie. equal time on and off)
' Thus, if you set the period to 30 milliseconds (as in this example),
' the signal will be high for 15 milliseconds and low
' for 15 milliseconds
period = 30000 ' 30 milliseconds
If wrapper.isFeatureSupportedContinuousStrobe(spectrometerIndex) = True Then
    continuousStrobeFeature =
        wrapper.getFeatureControllerContinuousStrobe(spectrometerIndex)
    ' Set the period of the continuous-strobe signal
```

```
   continuousStrobeFeature.setContinuousStrobeDelay(period) ' odd name!
Else
   MsgBox("continous-strobe not supported by this spectrometer")
   Return
End If
' Continuous-strobe signal will not be enabled until
' you turn the lamp on, so...
wrapper.setStrobeEnable(spectrometerIndex, 1)
' Lamp state doesn't change until we ask for a spectrum, so...
wrapper.getSpectrum(spectrometerIndex)
```

External Trigger Delay

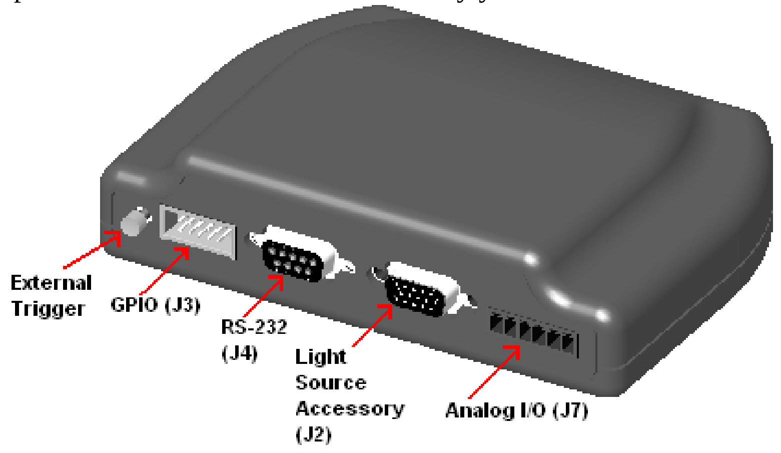
See your spectrometer's documentation for information about this feature. Additionally, the OmniDriver javadocs for the ExternalTriggerDelay class provide details on how to use External Trigger Delay.

GPIO (General Purpose I/O)

Using the GPIO feature, your software can exchange (send *and* receive) electrical signals with external devices. GPIO provides up to 10 separate bits for this purpose. Each bit may be individually configured as input or output. Response is immediate -- when you set the value of a GPIO output bit, the voltage changes immediately.

In the software, bits are numbered 0 to 9. Different spectrometer models may have varying numbers of GPIO bits available. See the documentation for your spectrometer.

The HR4 Breakout Box provides physical access to these GPIO bits. The J3 connector of this device has 10 pins which correspond to the 10 GPIO bits controlled by your software as shown below.



Rear Panel of HR4 Breakout Box

When facing the 10-pin J3 GPIO connector on the rear of the Breakout Box, bit numbering is as follows:

1	3	5	7	9
0	2	4	6	8

10-Pin J3 GPIO Connector Bit Numbering Diagram

Note

Speed limitation: due to hardware limitations in the spectrometer, OmniDriver must impose a 200 millisecond delay before returning from any of the GPIO function calls (e.g., `setDirectionBit(), setValueBit()`, etc.).

Visual Basic – COM interface

```
Dim gpioBitSet As OmniDriver.CCoBitSet
Dim gpioController As OmniDriver.CCoGPIO
Dim spectrometerIndex As Integer
spectrometerIndex = 0
' Always verify that the feature is supported by your spectrometer
If wrapper.isFeatureSupportedGPIO(spectrometerIndex) = False Then
   MsgBox("The GPIO feature is not supported by this spectrometer.")
   Return
End If
' The following call to getFeatureControllerGPIO() will always return a '
"valid" object, even if the spectrometer does not support that
' feature.
' Hence it is important to first call isFeatureSupportedGPIO()
gpioController = wrapper.getFeatureControllerGPIO(spectrometerIndex)
' Set the mode: false is normal GPIO mode;
'               true is "alternate function" mode
gpioController.setMuxBit(0, false)
gpioController.setMuxBit(1, false)
' Set the direction: true is output, false is input
gpioController.setDirectionBit(0, true) ' set bit 0 to output
gpioController.setDirectionBit(1, false) ' set bit 1 to input
' Read all 10 GPIO bits into our buffer, even if some of them were
' set to output (we will simply ignore those bits)
gpioBitSet = gpioController.getValueBits() ' read all GPIO bits
MsgBox("GPIO bit 1 has the value " & gpioBitSet.get(1))
gpioController.setValueBit(0, true) ' output: set GPIO bit 0 "high"
```

Visual Basic – "C" interface

```
Dim bitPosition As Long
Dim bitSetHandle As Long
Dim bitValue As Long
Dim gpioHandle As Long
Dim spectrometerIndex as Long

spectrometerIndex = 0
If Wrapper_isFeatureSupportedGPIO(wrapper, spectrometerIndex) = 1 Then
   gpioHandle = GPIO_Create()
   Wrapper_getFeatureControllerGPIO wrapper,spectrometerIndex,
            gpioHandle
   bitPosition = 0
   GPIO_setMuxBit gpioHandle, bitPosition, 0 ' use this bit for
                                            ' general purpose I/O
   GPIO_setDirectionBit gpioHandle, bitPosition, 1 ' set direction to
                                                   ' "output"
   GPIO_setValueBit gpioHandle, bitPosition, 1

   ' Demonstrate how to use the BitSet structure to read multible bits
   ' simultaneously
   GPIO_setMuxBit gpioHandle,0,0 ' use this bit for general purpose I/O
   GPIO_setMuxBit gpioHandle,1,0 ' use this bit for general purpose I/O
   GPIO_setMuxBit gpioHandle,2,0 ' use this bit for general purpose I/O
   GPIO_setMuxBit gpioHandle,3,0 ' use this bit for general purpose I/O
   GPIO_setMuxBit gpioHandle,4,0 ' use this bit for general purpose I/O
   GPIO_setMuxBit gpioHandle,5,0 ' use this bit for general purpose I/O
   GPIO_setMuxBit gpioHandle,6,0 ' use this bit for general purpose I/O
   GPIO_setMuxBit gpioHandle,7,0 ' use this bit for general purpose I/O
   GPIO_setMuxBit gpioHandle,8,0 ' use this bit for general purpose I/O
   GPIO_setMuxBit gpioHandle,9,0 ' use this bit for general purpose I/O
   GPIO_setDirectionBit gpioHandle, 0, 0 ' set direction to "input"
   GPIO_setDirectionBit gpioHandle, 1, 0 ' set direction to "input"
   GPIO_setDirectionBit gpioHandle, 2, 0 ' set direction to "input"
   GPIO_setDirectionBit gpioHandle, 3, 0 ' set direction to "input"
   GPIO_setDirectionBit gpioHandle, 4, 0 ' set direction to "input"
   GPIO_setDirectionBit gpioHandle, 5, 0 ' set direction to "input"
   GPIO_setDirectionBit gpioHandle, 6, 0 ' set direction to "input"
   GPIO_setDirectionBit gpioHandle, 7, 0 ' set direction to "input"
   GPIO_setDirectionBit gpioHandle, 8, 0 ' set direction to "input"
   GPIO_setDirectionBit gpioHandle, 9, 0 ' set direction to "input"
   bitSetHandle = BitSet_Create()
   listboxMessages.AddItem ("About to read GPIO bits")
   listboxMessages.Refresh
   GPIO_getValueBits gpioHandle, bitSetHandle ' read all 10 bits
   bitValue = BitSet_get(bitSetHandle, 0)
   listboxMessages.AddItem ("GPIO bit 0 = " & bitValue)
   bitValue = BitSet_get(bitSetHandle, 1)
   listboxMessages.AddItem ("GPIO bit 1 = " & bitValue)
   listboxMessages.Refresh
   BitSet_Destroy bitSetHandle ' don't forget to do this!

   GPIO_Destroy gpioHandle
End If
```

I2C Bus Communications Feature

Call `wrapper.isFeatureSupportedI2CBus(spectrometerIndex)` to determine if your spectrometer supports this feature.

Call `wrapper.getFeatureControllerI2CBus(spectrometerIndex)` to obtain a controller object for this feature.

```
if (wrapper.isFeatureSupportedI2CBus(spectrometerIndex) == true)
    {
        byte[] buffer = new byte[16];
        byte[] readBuffer = new byte[128];
        int length;

        System.out.println("I2CBus feature is supported");
```

```
            I2CBus i2cbus = wrapper.getFeatureControllerI2CBus(spectrometerIndex);
            try {
               buffer[0] = 0x00; // eeprom memory address MSB
               buffer[1] = 0x00; // eeprom memory address LSB
               buffer[2] = 0x05;           // value to write into eeprom at that address
               buffer[3] = 127;       // value to write into eeprom address+1
               buffer[4] = 0x70;           // value to write into eeprom address+2
               buffer[5] = (byte)128; // value to write into eeprom address+3
               wrapper.getWrapperExtensions().setI2CBytes(i2cbus,(byte)0x53,(byte)6,buffer); //
i2cBus_object, i2c_address, byteCount, payload

               buffer[0] = 0x00; // eeprom memory address MSB
               buffer[1] = 0x00; // eeprom memory address LSB
               // In the following statement, 0x53 is the address of the second EEPROM in the USB2000+
spectrometer
               // CAUTION: do NOT use this code to write into your spectrometer's EEPROM - you will
overwrite important data!
               wrapper.getWrapperExtensions().setI2CBytes(i2cbus,(byte)0x53,(byte)2,buffer); // i2c, address,
byteCount, buffer

               length = 4;
               readBuffer = wrapper.getWrapperExtensions().getI2CBytes(i2cbus,(byte)0x53,(byte)length); //
i2c, address, byteCount
               for (int index=0; index<length; ++index)
                  System.out.println(readBuffer[index]);

            } catch (IOException exception) {
               System.out.println("I2C problem: " + exception);
            }
         } else {
            System.out.println("I2CBus feature is not supported");
         }
```

To send bytes:

Call wrapper`.getWrapperExtensions().setI2CBytes(i2cbus, address, numberOfBytes, outputBuffer)` This method will return a "1" if the operation was successful. It will return a "0" if the operation failed.

Call `wrapper.getLastException()` to determine the nature of the error.

To receive bytes

Call `wrapper.getWrapperExtensions().getI2CBytes(i2cbus, address, numberOfBytes)`

If an error occurred, this method returns a single-element array with that element's value set to "-1". If successful, this method returns an array containing the bytes received from the device.

Note

It is possible to send and receive bytes by calling the `setI2Cbytes()` and `getI2Cbytes()` methods directly on the I2CBus controller object. However, this is NOT recommended because any I/O error will result in an application crash.

Irradiance Calibration Factor

See your spectrometer's documentation for information about this feature. Additionally, the OmniDriver javadocs for the IrradianceCalibrationFactor class provide details on how to use this feature.

Nonlinearity Correction

See your spectrometer's documentation for information about this feature. Additionally, the OmniDriver javadocs for the NonlinearityCorrectionProvider class provide details on how to use this feature.

SPI Bus

See your spectrometer's documentation for information about this feature. Additionally, the OmniDriver javadocs for the SPIBus class provide details on how to use this feature.

```
if (wrapper.isFeatureSupportedSPIBus(spectrometerIndex) == true)
    {
        int length;
        byte[] buffer = new byte[16];
        byte[] readBuffer;

        System.out.println("SPIBus feature is supported");

        SPIBus spibus = wrapper.getFeatureControllerSPIBus(spectrometerIndex);
        length = 7;
        buffer[0] = 17; // 0x11
        buffer[1] = 17; // 0x11
        buffer[2] = 34; // 0x22
        buffer[3] = 34; // 0x22
        buffer[4] = (byte)127;
        buffer[5] = (byte)128;
        buffer[6] = (byte)0x80;
        try {
            readBuffer = spibus.getSPIBytes(buffer, length);
            for (int index=0; index<readBuffer.length; ++index) {
                System.out.print("byte[" + index + "] = " + readBuffer[index] + "  ");
            }
            System.out.println();
        } catch (IOException exception) {
            System.out.println("SPI problem: " + exception);
        }
    } else {
        System.out.println("SPIBus feature is not supported");
    }
```

Single Strobe

The Single Strobe feature allows you to pulse a lamp for a specified duration at a fixed offset from the start of the integration period. This pulse will be repeated once for each integration period. The Single Strobe mode is enabled by setting the delay time and pulse duration parameters and then calling `wrapper.setStrobeEnable(true).`

The Single Strobe feature transmits an electrical signal on one of the output pins of the spectrometer. Your lamp plugs into this connector on the spectrometer and is controlled by the state of the Single Strobe pin. See your spectrometer documentation to determine which pin carries the Single Strobe signal.

Four separate connector pins relate to lamp control. These signals are as follows:

1. Strobe Enable signal (typically pin 4*)
2. Single Strobe signal (typically pin 8*)
3. Continuous Strobe signal (typically pin 5*)
4. Ground (typically pin 6*)

* Check your spectrometer's documentation to verify the pin numbers.

To set the length of delay from the start of the integration period before turning the lamp on, call `singleStrobe.`**`setSingleStrobeHigh`**`(int highDelay)`, passing in a value of microseconds, in the range 0 to 65535.

To specify when the lamp should be turned off, call `singleStrobe.`**`singleStrobeLow`**`(int LowDelay)`. The value you specify here is the number of microseconds (0 to 65535), relative to the start of the integration period, when you want the lamp to be turned off again. The actual duration of the light pulse will be (`lowDelay` – `highDelay`), so the `lowDelay` value should be larger than the `highDelay` value. To send no pulse at all out the single-strobe signal pin, set the `lowDelay` value equal to the `highDelay` value.

Note

You will *not* see any signals/pulses on the single-strobe pin until you have called `wrapper.setStrobeEnable(true)`. You can call `wrapper.setStrobeEnable(true)` before or after setting the single-strobe delay/duration parameters; it makes no difference to the software. However, it is cleaner if you leave the lamp turned off (`wrapper.setStrobeEnable(false)`) until you have set up your single-strobe delay/duration parameters.

Caution

- **Not all spectrometers support the Single Strobe mode of operation. Check your spectrometer's documentation to verify that it supports Single Strobe mode.**
- **Not all lamps support Single Strobe mode. Check your lamp's documentation to verify the availability of this feature.**
- **Even if both the spectrometer and the lamp support Single Strobe operation, the lamp may not support setting the *duration* of the pulse. Strobe lamps such as the PX2 simply provide a strobe flash, regardless of the duration you specify. With these lamps, you can only control the delay until the flash occurs, not the duration of the flash.**

For further information, see Continuous Strobe.

Visual Basic

```
Dim singleStrobeFeature As OmniDriver.CCoSingleStrobe
Dim delayUntilStrobeOff As Integer
Dim delayUntilStrobeOn As Integer
Dim spectrometerIndex As Integer
spectrometerIndex = 0 ' arbitrarily choose the first spectrometer
' The following two constants will establish a pulse duration of
' 20 milliseconds (delayUntilStrobeOff - delayUntilStrobeOn)
delayUntilStrobeOn = 30000 ' 30 milliseconds
delayUntilStrobeOff = 50000 ' 50 milliseconds
If wrapper.isFeatureSupportedSingleStrobe(spectrometerIndex) = True Then
   singleStrobeFeature =
      wrapper.getFeatureControllerSingleStrobe(spectrometerIndex)
   ' Set the number of microseconds to delay after the
   ' start of the integration period before we turn the lamp on
   singleStrobeFeature.setSingleStrobeHigh(delayUntilStrobeOn)
   ' Set the number of microseconds to delay after the
   ' start of the integration period before we turn the lamp off
   singleStrobeFeature.setSingleStrobeLow(delayUntilStrobeOff)
Else
   MsgBox("single strobe feature not supported by this spectrometer")
   Return
End If
' Single-strobe signal will not be enabled until
' you turn the lamp on, so...
wrapper.setStrobeEnable(spectrometerIndex, 1)
' Lamp state doesn’t change until we ask for a spectrum, so...
wrapper.getSpectrum(spectrometerIndex)
```

Stray Light Correction

For information about this feature, see the documentation for your spectrometer. Additionally, the OmniDriver javadocs for the `StrayLightCorrection` class provide details on how to use this feature.

Thermo Electric Cooling

Some spectrometers such as the QE65000 have a thermo electric cooled (TEC) CCD that can be controlled by the software.

Be careful to observe the allowable temperature range supported by your spectrometer’s TE cooler. For example, the TE cooler of the QE65000 is capable of dropping the temperature of the CCD by 30-43 degrees Celsius below the ambient temperature. Thus, if the ambient temperature happens to be 25 degrees Celsius, the range of values you may pass in to the setDetectorSetPointCelsius() method will be -5 to -18 degrees Celsius. In this example, if you (erroneously) attempt to specify a value of -3 degrees Celsius, the TE cooler will not function properly and the temperature of the CCD will not approach -3 degrees Celsius.

Caution

You may see erroneous temperature values returned at random intervals by the `getDetectorTemperatureCelsius()` method. Typically, you will see just one bad value, and then the next call to this method returns the correct value.

Visual Basic – COM interface

```
Dim actualTemperature As Double
Dim desiredTemperature As Double
Dim tecController As OmniDriver.CCoThermoElectric
Dim spectrometerIndex As Integer

spectrometerIndex = 0 ' arbitrarily select the first spectrometer
' Attempt to obtain a "CCoThermoElectric" object that will allow us to
' interact with the  thermo-electric controls for this spectrometer.
If wrapper.isFeatureSupportedThermoElectric(spectrometerIndex) = False Then
   MsgBox("The thermo-electric feature is not supported.")
   Return
End If
' The following call to getFeatureControllerThermoElectric() will
' always return a "valid" object,
' even if the spectrometer does not support that feature.
' Hence it is important to first call sFeatureSupportedThermoElectric()
tecController =
   wrapper.getFeatureControllerThermoElectric(spectrometerIndex)

actualTemperature = tecController.getDetectorTemperatureCelsius()

' If you want to control the temperature, setTECEnable()
' MUST be set to true
tecController.setTECEnable(True)
tecController.setFanEnable(True) ' turn the fan on (optional)
desiredTemperature = -12.3 ' degrees Celsius
tecController.setDetectorSetPointCelsius(desiredTemperature)
```

Version

See the OmniDriver javadocs for the Version class for details about this feature.

Chapter 6

External Trigger Modes

Overview

The trigger mode setting of your spectrometer gives you more precise control over the timing of a spectrum capture.

Not all spectrometers support all trigger modes, so be sure to see the documentation for your particular spectrometer. You set the trigger mode of your spectrometer by calling `wrapper.setExternalTriggerMode()`, specifying an integer that corresponds to the desired mode.

Mode 0: Normal Mode

Sometimes called "free running" mode, this is the default mode for all spectrometers. In this mode, the spectrometer is continuously acquiring new spectra, using default power-up settings for integration time, etc.

The integration time is controlled by calls to the `wrapper.setIntegrationTime()` method.

When you call the `getSpectrum()` method, it returns the next available spectrum, subject to delay due to the length of integration time, number of scans to average, and possibly a stability scan. It is also possible that `getSpectum()` returns almost immediately, even if you specified a lengthy integration period, because the spectrometer is continuously acquiring spectra. See Appendix A: *FAQs* for more information.

The `getSpectrum()` method will block (i.e., not return) until the spectrometer finishes acquiring the current spectrum.

Mode 1: External Software Trigger Mode

In this mode, the trigger signal acts like an "enable". As long as the input trigger signal pin on your spectrometer is held electrically high, spectra will be continuously acquired. When the signal goes low, the spectrometer no longer acquires spectra, and `getSpectrum()` will remain blocked (i.e., not return) until the signal goes high again.

This mode is similar to Mode 0 (Normal mode) in that as long as the trigger signal remains high, the spectrometer is continuously acquiring new spectra, irrespective of when you call `getSpectrum()`. In this situation, `getSpectrum()` simply returns the next available spectrum.

The integration time is controlled by calls to the `wrapper.setIntegrationTime()` method.

Mode 2: External Synchronization Trigger Mode

In this mode, spectra acquisition is initiated by an external synchronizing TTL trigger signal. The purpose of this mode is to allow multiple spectrometers to be synchronized in terms of the start of their acquisition period, and the duration of the integration period. The integration time is determined based on an average of the length of time between the input trigger signal pulses. See your spectrometer's documentation for the allowed frequency range for this input signal.

Mode 3: Hardware Trigger Mode

In this mode, the spectrometer does not begin to acquire a new spectrum until the rising edge of an external TTL input signal. When you call `getSpectrum()`, this method blocks (i.e., does not return to the caller) until the trigger signal occurs and the spectrum has been acquired.

The integration time is controlled by calling the `wrapper.setIntegrationTime()` method.

Unique behavior of the QE65000: the CCD detector on the QE65000 does not support Stand-by mode. It must be running at all times. Therefore, the QE65000 simply sets its integration time to the minimum possible value (8 milliseconds) and waits for the trigger signal. When the signal occurs, it waits for the current acquisition to complete (can take up to 8 milliseconds), sets the integration time to the value specified by the user, and then acquires a new spectrum.

Note

Once you put the spectrometer in this mode, it is impossible to programmatically return to normal Hardware Trigger mode. You must power-cycle the spectrometer by unplugging it from your computer.

Mode 4: Single-Shot Trigger Mode (Quasi Realtime Mode)

This mode is designed to provide more precise software control over when a spectrum acquisition is initiated. This is especially valuable when you need to use very long integration periods, but also want precise control over when the acquisition period begins.

This mode is only available on certain spectrometers, including the QE65000, NIR256, and NIR512 spectrometers.

In this mode, the spectrometer automatically sets its integration time to a very short integration period and then begins to continuously acquire spectra using this minimal integration time. When `setIntegrationTime()` is called to specify the desired integration period, this value is stored in the spectrometer, but the spectrometer continues to acquire spectra using the minimal integration period.

When `getSpectrum()` is called, the spectrometer completes the current acquisition (which should happen very quickly since the integration time is so short). The spectrometer then sets its integration time to the value requested and initiates a new spectrum acquisition. When this acquisition completes, `getSpectrum()` returns the new spectrum. The spectrometer once again reverts to the minimal integration period and continues to acquire spectra in the background.

Chapter 7

High-Speed Data Acquisition

Overview

For applications that must acquire spectra as rapidly as possible, OmniDriver offers a High-speed Data Acquisition mode of operation. This mode is available for all Ocean Optics spectrometers. OmniDriver achieves this improved performance by pre-allocating buffer space for a fixed number of spectra. Speed is also improved since all spectra are acquired by a single call to a start-acquisition method, rather than using separate function calls for each spectrum. This avoids the overhead of having to pass down through several layers of software for each spectrum. It also avoids speed limitations imposed by the language in which you develop your application (e.g., LabVIEW, Visual Basic, etc.).

High-speed data acquisition mode requirements include the following:

- You must specify *in advance* the number of spectra to be acquired. A single call into the high-speed API will automatically acquire this number of spectra. Note that you are free to repeat the call to the high-speed acquisition API multiple times, each time acquiring a fresh set of spectra.
- Since buffers must be pre-allocated, the amount of memory allocated to the Java virtual machine limits you to about 3000 spectra.
- High speed data acquisition mode may be used for only *one* spectrometer at a time.

All methods comprising the high-speed data acquisition API belong to the **wrapper** object. These methods all begin with the prefix `highSpdAcq`.

Function Call Sequence

The sequence of function calls you must make to use High-speed Data Acquisition mode is as follows.

1. highSpdAcq_AllocateBuffer()
2. `highSpdAcq_StartAcquisition()`
   This method normally will not return until the requested number of spectra have been acquired. It may return before all requested spectra have been acquired if you call `highSpdAcq_StopAcquisition()` from another thread in your application.
3. highSpdAcq_GetNumberOfSpectraAcquired()
   Always call this method once acquisition has completed, just in case the number of spectra is less than expected.
4. Loop once for each spectrum acquired, calling the following methods.
   a. `highSpdAcq_GetSpectrum(spectrumNumber)`

b. `highSpdAcq_GetTimeStamp(spectrumNumber)`
This method is optional and should only be called if you require an exact time stamp indicating when each individual spectrum was acquired. This time stamp is an instance of the `HighResTimeStamp` object, defined in OmniDriver.

c. `highSpdAcq_IsSaturated(spectrumNumber)`
It is always a good idea to call this method on behalf of each acquired spectrum, to ensure that saturation did not occur.

One additional method that may be useful is `highSpdAcq_StopAcquisition().` You normally do not need to call this method since acquisition of spectra automatically stops when the required number of spectra has been acquired. However, if you detect a condition which obviates the need for acquiring all of the requested spectra, you may call this method. This method must be called from a different thread in your application since the highSpdAcq_StartAcquisition() method blocks until it has acquired all requested spectra.

For further details about each of the high-speed data acquisition API methods, refer to Chapter 11: *Wrapper API Reference*.

You may also want to take a look at the `HighSpeedAcquisitionSample`, written in Java, to see high-speed data acquisition in action.

Notes

If your application runs out of memory, see **Appendix D: Controlling Memory Size**.

For ideas on how to improve the performance of your application, see **Appendix E: Improving OmniDriver Performance on Windows**.

Chapter 8

Using OmniDriver with Multithreaded Applications

Overview

OmniDriver supports multi-threaded applications. Using multiple threads, your application can acquire and control spectra from multiple spectrometers, all running concurrently, without interfering with each other.

You can only run one OmniDriver application at a time on your PC. When your application calls `wrapper.openAllSpectrometers()`, OmniDriver attempts to identify all attached spectrometers. OmniDriver then assumes it has exclusive access to each of the spectrometers it discovers. If you were to run a second OmniDriver application on the same PC, both applications would assume they had exclusive control of the same spectrometer, and they would interfere with each other.

Notes

The C, C++, and .NET assembly interfaces all support true multithreading/concurrency. Avoid the deprecated COM interface because it does not support multithreading.

If your application runs out of memory, see **Appendix D: Controlling Memory Size**. Multithreading General Guidelines

Observe the following guidelines when performing multithreading in an OmniDriver application:

- There should only be ONE instance of the Wrapper object. Usually you will instantiate/create this object in your application's main thread.
- `The wrapper.openAllSpectrometers()` function should only be called *once*, probably in your main thread.
- When you create a new thread, you will want to pass in a reference to the one-and-only Wrapper object. In C++ or Java, you could pass this reference in via the constructor for your new thread.
- Each spectrometer should be accessed from only one thread. Never try to invoke Wrapper methods for the same spectrometer in two different threads. Results are unpredictable and the application will probably crash, or give incorrect results without reporting an error. You can access more than one spectrometer from a single thread, but these particular spectrometers must always, and only, be accessed from that thread.

- Be aware that Windows, Linux and Mac OSX are not real-time operating systems (and Java further complicates the issue). You can't predict how Windows/Java will allocate CPU cycles to any given thread. You are, however, free to alter thread priority using whatever function calls your language provides you. You are also free to invoke sleep() and yield() methods in your threads. (These may not be the exact names of these methods for your IDE/language.)
- The `wrapper.getSpectrum()` method blocks (i.e., does not return) until a new spectrum is available. When you call `wrapper.getSpectrum()` in one thread, it only blocks that thread. It does not impact any other thread.
- In general, if a wrapper method does not take a `spectrometerIndex` parameter, you should call that method only from the main thread of your application. If a wrapper method *does* take a `spectrometerIndex` parameter, you should only call that method from the thread that "owns" that spectrometer.

Exceptions to General Guidelines

The following are exceptions to the guidelines listed above. These actions are permitted, but not recommended due to their complexity and risk of confusion.

- Actually it is perfectly safe to access the same spectrometer from multiple threads **if you can guarantee that the threads will not attempt to call functions that access the same spectrometer at the same time**. It is much safer and easier to manage your application if you restrict a given spectrometer to a single thread, but we do not enforce any restriction on accessing the same spectrometer from different threads. A problem will arise only if you attempt to call functions that access the same spectrometer at the same time from two different threads. If you are certain that your threads are coordinated in such a way that this will never happen, then you should have no problem.
- You may access all of your spectrometers from the main thread prior to creating any background threads. Again, the only absolutely incontrovertible restriction is that you must not attempt to call functions that access a given spectrometer from two different threads at the same time.

ADC1000-USB A/D Converter Notes

The ADC1000-USB A/D Converter supports multiple channels. You might think that each channel can be controlled via a separate thread. However, due to the design of the ADC1000-USB hardware, all of its channels *must* be accessed from the same thread. The channels of the ADC1000-USB do not operate like independent spectrometers.

LabVIEW Notes

The LabVIEW call library functions that call functions in OmniDriver32.dll must be made to be multi-threading. To do this, perform the following procedure.

► ***Procedure***

1. Double click on the Call Library function on the back panel.

For LabView 8.5, select the Function tab and find the top right section called **Thread**. There are two options: **run in UI thread** and **run in any thread**. Select **run in any thread**.

Earlier versions of LabVIEW have a drop-down box near the upper right corner with the options **Run in UI thread** and **Reentrant**. Select **Reentrant**.

This must be done for all call library functions in the VI.

2. Make that sub-VI reentrant. To do this:
    a. From the **File** menu option, select **VI properties**. There is a **Category** drop down box.
    b. Select **execution**.
    c. Locate the **Reentrant** check box (on the left, about midway down). Check this box.
    d. Locate the **Allow debugging** check box. This needs to be unchecked.

The sub VI should now be fully multithreading.

Samples

We provide samples, written in Java and C# (C-Sharp) and Visual Basic (2005), that illustrate how to do multi-threading with OmniDriver:

- Java: Sample is located in the ..\samples\java\ ConcurrentAcquisitionThreads directory.
- C#: Sample is located in the ..\samples\win32\csharp\ ConcurrentAcquisitionThreads directory.
- Visual Basic: Sample is located in ..\samples\win32\visualbasic\ vs2005_com_MultiThreading.

Chapter 9

Developing Your OmniDriver Application

Overview

The following sections provide information on setting up the development environment for your specific programming language to develop OmniDriver applications.

Note

The source code and the project files for all sample applications are available in a separate downloadable installer located on the Ocean Optics website: **Windows OmniDriver and SPAM Sample Pack**.

NetBeans Setup on Linux

If you are using the NetBeans IDE on Linux, use the following guidelines for setting up your environment.

2. Install the latest version of OmniDriver on your Linux system. Be sure to choose the correct architecture (i.e., 32-bit or 64-bit). For this example, assume your system is 64-bit architecture, so the installer would be similar to the folowing:
   OmniDriver-2.12-linux64-development-installer.bin

   This procedure assumes you install OmniDriver into your home directory.
3. In NetBeans, open the sample project located in ~/OceanOptics/OmniDriver-2.12/samples/java/SpectrometerTest
4. You may need to resolved references in NetBeans. To do this:
   a. In the project window, right-click on the SpectrometerTest project and choose **Resolve Reference Problems…**
   b. Highlight the problem JAR file in the list and click **Resolve**.
   c. Navigate to ~/OceanOptics/OmniDriver-2.12/OOI_HOME and highlight the missing JAR file. Then click **OK**.

5. You may need to add OmniDriver.jar to the list of "Compile-time Libraries" used by this project. To do this:
    a. Right-click on the SpectrometerTest project and choose **Properties**.
    b. Click on **Libraries** in the **Categories** list.
    c. Click on the **Compile** tab.
    d. Click the **Add Jar/Folder** button.
    e. Navigate to the OOI_HOME folder, highlight OmniDriver.jar, and then click **OK**.
6. Build your app.
7. Try running your app.
8. If you get the error "Failed to load any native library for USB…", do the following:
    a. Right-click on the project and choose **Properties**.
    b. Click on **Run** in the list of categories.
    c. In the **VM Options** field, enter something like: -Djava.library.path=/home/steve/OmniDriver-2.12/OOI_HOME (or where you installed OmniDriver)
9. Rebuild and run the sample application to verify your environment has been set up correctly.

Developing in C/C++ (all IDEs)

Required Header Files

If you are developing in C or C++, you will need several header files.

You must update your IDE's project settings by adding the following directories to the list of include directories:

- `$(OMNIDRIVER_HOME)\include`
- `$(JAVA_HOME)\include`
- `$(JAVA_HOME)\include\win32`

`OMNIDRIVER_HOME` refers to the location where OmniDriver was installed.
`JAVA_HOME` refers to the location where the Java JDK was installed.

Adding Project Directories in Microsoft Visual Studio 2005

► ***Procedure***

To add project directories in Microsoft Visual Studio 2005,

1. Open the project Properties window.
2. Expand the C/C++ branch, and click **General**.
3. Add each of the directories listed above to the **Additional Include Directories** property.

Required Library Files

You must also update your IDE's project settings for it to locate the OmniDriver32.lib, common32.lib, and (optionally) SPAM32.lib files.

► ***Procedure***

1. In Microsoft Visual Studio 2005, open the project Properties window, expand the **Linker** branch, and click on **General**.
2. Add the full path to the OOI_HOME directory to the list of directories for the **Additional Library Directories** property.
3. Click on the **Input** item for linker settings and add OmniDriver32.lib, common32.lib, and (optionally) SPAM32.lib to the **Additional Dependencies** property.

Java Requirement

As implied above, you must download and install the Java JDK. The JDK version of Java is necessary because the smaller JRE version does not have the necessary include files. Any version of Java 1.6 or later is acceptable.

Download the Java JDK from **www.javasoft.com** .

Specify Your OS Platform

It may be necessary to add one of the following #define lines to your source files:

- `#define WIN32` (use this for Windows 64-bit also)
- `#define MACOSX`
- `#define LINUX`

Choose the symbol that corresponds to the operating system you are using. This #define should be placed BEFORE any include statements for OmniDriver header files. Some IDE's and project types automatically define the appropriate symbol. But some types of Visual Studio projects do not automatically define the required symbol. If none of these symbols is defined, you will get numerous compile-time errors relating to OmniDriver symbols.

Interface Choices

OmniDriver functionality is exposed via three different interfaces:

- C functions
- C++ classes
- COM objects deprecated
- .NET assemblies

If you are developing with Microsoft Visual Studio, you may use any of these interface methods. However, if you are developing with any other IDE (e.g., Borland C++ Builder), you may only use the C and COM interfaces. This is because there is no standard for how C++ name mangling is performed, and the OmniDriver C++ interface was developed using Microsoft Visual Studio. Using the C++ interface with an IDE other than Visual Studio results in many "undefined symbol" errors.

Developing with Borland C++ Builder

When using the Borland IDE, you must use the C interface to OmniDriver due to incompatibilities in how Borland performs C++ name mangling vs. Microsoft Visual Studio. You also have the option of using the OmniDriver COM object interface, but that is outside the scope of this document.

Setting Up the Environment

The format of the library (*.lib) files provided by OmniDriver is incompatible with the Borland C++ Builder IDE. To overcome this, you must manually generate your own *.lib files based on the OmniDriver DLL files.

Note

Be careful not to over-write the existing *.lib files in the OOI_HOME directory in case you need the Microsoft form of these files in the future.

► ***Procedure***

1. Open a DOS window
2. cd into the OOI_HOME directory
   (c:\program files\ocean optics\OmniDriverSPAM\OOI_HOME)
3. Use the Borland implib.exe tool to generate new library files. This tool may be located in c:\program files\borland\bds\4.0\bin.
   In the following commands, be careful to specify an output lib filename that will not overwrite the existing *.lib files already located in the OOI_HOME directory. In the example commands below for 32-bit systems, "_borland" was appended to the names of the output files to ensure they will not over-write other library files (for 64-bit, replace "32" with "64"):

```
a. implib  common32_borland.lib common32.dll
b. implib  OmniDriver32_borland.lib OmniDriver32.dll
c. implib  SPAM32_borland.lib SPAM32.dll
```

4. Edit the file platform.h in the OmniDriver include subdirectory.
5. Find the **#ifdef __BORLANDC__** statement.

   The following line should look like

```
#define EXPORTED __declspec(dllexport) __stdcall
```

6. Delete **the __stdall** keyword from this line and save the file back to disk.

Creating a New Project

► ***Procedure***

1. Start Borland C++ Builder using **File | New | Other.**
2. Highlight **C++ Builder Projects** in the left pane.
3. Highlight **VCL Forms Application** in the right pane.
4. Click **OK**.

Note

For convenience, you can make the Properties pane visible by clicking **View.ObjectInspector**. This is handy for setting the text labels and control names for various GUI components.

---

5. Inform Borland of the location of the required header files as follows:

Select Project | Options.

Expand the C++ Compiler node.

Highlight Paths and Defines.

Click the **Edit** button on the **Include search path line** to add the following pathnames (these paths may be slightly different on your system):

c:\program files\ocean optics\OmniDriver\include

6. Inform Borland of the location of the dll/library files as follows:
   a. Select Project | Options.
   b. Expand the C++ Compiler node.
   c. Highlight Paths and Defines.
   d. Add **c:\program files\ocean optics\OmniDriver\OOI_HOME**

7. Tell Borland which OmniDriver library files (*.lib) to link with as follows:

Select Project | Add To Project..

Set Files of type to Library file (*.lib).

Navigate to the OOI_HOME directory and add the following files:

```
common32_borland.lib
OmniDriver32_borland.lib
SPAM32_borland.lib
```

Caution

For each “C” callable function in OmniDriver, we provide support for two different calling conventions. Borland requires the **stdcall** alternative. Therefore, you must append **_stdcall** to every OmniDriver function call in your application. This ensures that your application links to and calls the correct form of the function.

Troubleshooting

If you forget to edit the platform.h file, or if you reinstall OmniDriver (thus over-writing your edits) you will see error messages like ***E2138 Conflicting type modifiers***.

If you forget to append **_stdcall** to one of your OmniDriver function calls, you will get ***Error: Unresolved external*** error messages. The solution is to edit your source file to append **_stdcall** to each OmniDriver function call.

You may see warning messages like ***W8017 Redefinition of _ASSERTE is not identical***. These warnings are not a problem and may safely be ignored.

Developing in C with the National Instruments CVI C Compiler

Use the following procedure to set up for new projects with OmniDriver.

► ***Procedure***

Caution

STEP 1 IS IMPORTANT. If you fail to do this, you will get Missing Prototypes error messages when you attempt to compile applications using OmniDriver.

1. Within the CVI IDE, you must make the following changes:
   a. Select **Build Options** from the Options menu.
   b. Uncheck **Require function prototypes**.
2. Add the following header file directories to your CVI project:
   - $(OMNIDRIVER_HOME)\include
   - $(JAVA_HOME)\include
   - $(JAVA_HOME)\include\win32

   To do this:
   a. Click on the **Edit.Project** menu item
   b. Click the **Include Paths** button.

   c. Add each of the required include directories.

3. Add the following library files to your CVI project:

   - common32.lib
   - OmniDriver32.lib
   - SPAM32.lib (optional)

   These library files are located in the OmniDriver\OOI_HOME directory.
   To do this:

   a. Click on the **Edit.Project** menu item.
   b. Remove the three LIB files listed above in this step from the list if they are present. (highlight each file and click **Remove**).
   c. Click **Add** to re-add each of these three LIB (not DLL) files.

Developing with Microsoft Visual C++ 6.0

Specifying the Location of the Java Header Files

You must specify where to find the Java header files.

► ***Procedure***

To set the path where the IDE will look for the Java header files:

1. Click on the **Project.Settings** menu item.
2. Select **All Configurations** from the **Settings For** drop down list box.
3. Click on the **Source Files** in the pane on the left side.
4. Click on the **C++** tab in the pane on the right side.
5. Choose **Preprocessor** from the **Category** drop down list box.
6. There are two paths that reference Java . Modify both Java paths to correspond to the location where Java has been installed on your system.

Specifying the Location of the OmniDriver32.dll

You must specify where to find the OmniDriver32.dll.

► ***Procedure***

To set the path where the IDE will look for the OmniDriver32.dll:

1. Click on the **Project.Settings** menu item.
2. Select **All Configurations** from the **Settings For** drop down list box.
3. Click on the **Link** tab.
4. Choose **Input** from the **Category** drop down list box.

5. Enter C:\Program Files\OceanOptics\OmniDriver\OOI_HOME in the Additional library path textbox.

Handling the "long long" Data Type Issue

This compiler does not support the newer "long long" language construct for declaring 64-bit integers. To work around this problem, add the following two statements to the top of every *.cpp file in your project (ahead of any other #includes of OceanOptics header files):

- `#define HIGHRESTIMESTAMP_H`
- `#define SPECTROMETERCHANNEL_H`

Adding these two statements will prevent the inclusion of header files containing the "long long" data type. The excluded header files are not necessary if you are using the Wrapper API to control your spectrometers.

Developing in C#

PREFERRED: To use the .NET assembly interface to OmniDriver in your C# application, you must add a reference to the assembly as follows:

► ***Procedure***

1. In your application, click on the **Project** menu item, and then choose **Add Reference**.
2. Click on the **Browse** tab.
3. Navigate to the OOI_HOME directory.
4. Highlight **NETOmniDriver.dll** and/or **NETSpam.dll** and click **OK**.

Note

The ConcurrentAcquisitionThreads sample in the samples\win32\csharp directory demonstrates how to use the C calling interface to use OmniDriver within your C# application.

Creating a New C# VS2005 Project that Uses the .NET Interface to OmniDriver

► ***Procedure***

1. Create new project of type C# “Windows Application”.
2. Add a reference to NETOmniDriver64.dll and/or NETSpam64.dll:
   a. In Solution Explorer, right-click on **References** and choose **Add Reference…**
   b. Click the **Browse** tab.
   c. Navigate to the OOI_HOME folder.
   d. Highlight **NETOmniDriver.dll** and click **OK**.
   e. Repeat Step d for NETSpam.dll if you are using any SPAM functions.

Sample source code that accesses OmniDriver via the .NET interface:

```
int numberOfSpectrometers;
int spectrometerIndex;
OmniDriver.NETWrapper wrapper = new OmniDriver.NETWrapper();
numberOfSpectrometers = wrapper.openAllSpectrometers();
listBoxMessages.Items.Add("number of spectrometers = " +
numberOfSpectrometers);
if (numberOfSpectrometers < 1)
      return;
spectrometerIndex = 0;
listBoxMessages.Items.Add("serial number = " +
wrapper.getSerialNumber(spectrometerIndex));
```

Deploying Your C# Application

Normally, all that is needed to deploy your C# application is the EXE file containing the application itself. And you will also need to deploy the appropriate OmniDriver “redistributable” installer (no password required by this installer).

However, if your C# application uses the OmniDriver .NET assembly interface, you must also ensure that a copy of NETOmniDriver.dll and/or NETSpam.dll is placed in the same folder as your application’s EXE file. You can obtain these two DLL files from the OOI_HOME directory.

Developing in Delphi

The procedure in this section is based on Borland Developer Studio 2006. It will explain how to use OmniDriver’s COM interface in your Delphi application.

► ***Procedure***

1. Start Delphi for Microsoft Win32.
2. Click on the menu item **File.New – VCL Forms Application – Delphi for Win32**.

.

3. Do a File.SaveAll to a folder like “C:\MyApp\application”.
4. Click **File.SaveAll**. Specify something like “C:\MyApp\package”.
5. Click **Component.ImportComponent**.
    a. Type of component: Choose **Import a Type Library.** A list of registered type libraries will be displayed.
    b. Highlight **OmniDriver.NET interface.** Be sure to double-check the Filename path to ensure that you are selecting the correct version of OmniDriver.NET interface. The path should be something like C:\Program Files\Ocean Optics\OmniDriver \OOI_HOME\NETOmniDriver.tlb . Then click **Next**.

Note

If OmniDriver.NET interface is not available, click the **Add** button, navigate to the OOI_HOME directory where you installed OmniDriver, highlight **NETOmniDriver.tlb** and click **OK**. Then click **Next**.

---

6. On the Component panel:
    a. On the Palette page, choose **ActiveX**.
    b. Unit Dir Name: Change the path to the folder you just specified when you clicked **File.SaveAll** (in Step 3 above), e.g. C:\My App\application.
    c. Leave all other settings unchanged.
7. On the Create Unit panel, choose **Add unit to NETOmniDriver.bdsproj project** and click **Finish**.
8. In the Uses section of the Unit1.pas source file, add the **NETOmniDriver_TLB** and **ActiveX** namespaces.
9. In your application’s “var” section, declare an instance of “wrapper” as follows:

   wrapper: TNETWrapper;.

10. Double-click on your application’s main dialog box to generate the FormCreate method. This is where you should add the following OTO logic to initialize OmniDriver:
    wrapper := TNETWrapper.Create(self);
    wrapper.CreateWrapper();

Note

Refer to the sample named “DelphiSample” for a complete example showing how to control the spectrometer and acquire spectra. This sample is located in C:\Ocean Optics\OmniDriverSamplePack\omnidriver\win32\delphi

---

To declare the wrapper object:

```
   var
wrapper: TCCoWrapper;
```

Statements needed to initialize OmniDriver in your application:

```
   wrapper := TCCoWrapper.Create(self);
wrapper.CreateWrapper();
numberOfSpectrometers := wrapper.openAllSpectrometers();
```

Note

When you run your application, you may be prompted for the location of the file NETOmniDriver_TLB.pas. This file is located in the directory containing the OmniDriver "package" you created earlier.

Developing in Java

If you are using Java to develop applications using OmniDriver, you must install the Java JDK (*not* JRE) version 1.6 or above.

Update your system PATH to point to the new java location (for example, C:\Program Files\Java\JDK1.6.0_10). If Java is not in the system PATH, it uses the following registry key to attempt to locate java: `HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft\Java Runtime Environment\CurrentVersion`.

Troubleshooting Tips

If you run your OmniDriver-based Java application and get the exception NoClassDefFoundError for the com.oceanoptics.omnidriver.Wrapper class it is probably because the OOI_HOME environment variable has not been defined.

If you get the following exception when you run your OmniDriver-base Java application: **UnsatisfiedLinkError no NatUSB in java.library.path**, it is probably because the OOI_HOME directory is not in your system PATH or LD_LIBRARY_PATH on Linux.

Developing in LabVIEW

OmniDriver now provides a .NET 2.0 interface. We recommend that all LabVIEW applications use the .NET 2.0 interface when accessing OmniDriver functions. The VI interface to OmniDriver has been deprecated.

Note

LabVIEW versions before 2012 do not appear to fully support .NET 4.0. Based on our experience, LabVIEW developers can only use .NET 4.0 when developing within the LabVIEW IDE. If using LabVIEW prior to 2012, you MUST use the .NET 2.0 interface to OmniDriver if you are developing LabVIEW applications. For a fuller discussion of this limitation, see http://digital.ni.com/public.nsf/allkb/32B0BA28A72AA87D8625782600737DE9

The directory path C:\Program Files\OceanOptics\OmniDriver\labview\win32\VersionNN.MM contains the *.llb libraries that wrap the functions in the OmniDriver32.dll and SPAM32.dll files. Be sure to use the version of the Omnidriver LLB files that correspond to your version of LabVIEW.

OmniDriver32.dll, SPAM32.dll, and common32.dll are located, by default, in the directory named C:\Program Files\OceanOptics\OmniDriver\OOI_HOME. The OmniDriver sample applications that work with LabVIEW must know the location of these DLL files. If LabVIEW cannot find the OmniDriver DLLs, you will be asked for their location when you start the sample program.

Notes

The Wrapper library contains most of the basic spectrometer control functions. You should only use the functions/VIs belonging to the Wrapper library object. These functions comprise the Wrapper API. The only time you may use functions/VIs outside of the Wrapper is if one of the wrapper VIs itself returns an object of a different type. The following is a list of these other objects/libraries that correspond to features available on some, but not all, of the Ocean optics spectrometer models:

- AnalogIn
- AnalogOut
- BoardTemperature
- ContinuousStrobeDelay - (LabVIEW 8.x versions only)
- ExternalTriggerDelay - (Limited features available in LabVIEW 7.1)
- GPIO
- Single Strobe
- ThermoElectric

Important: the VI interface provided by OmniDriver has been deprecated. Please use the .NET interface to access OmniDriver from your LabVIEW application.

Cautions

For users of LabVIEW version 8.6: you must not install the FieldPoint Drivers option. This LabVIEW option prevents OmniDriver applications from running. If this option is installed, you will get the error message “Can’t create JVM” followed by LabVIEW error code 1097. The only solution is to uninstall the FieldPoint Drivers option.

Developing in VBA

The following sections provide information on how to write a “Visual Basic for Applications” macro to access OmniDriver functionality.

Writing a VBA Function

► ***Procedure***

1. Start Excel and open a new workbook.
2. From within Excel, click on the **Tools | Macro | Visual Basic Editor** menu item. This opens the Visual Basic Editor window.
3. From within the Visual Basic Editor, click on the Insert | Module menu item. This opens an empty window titled, for example, **Module1 (Code)**.
4. Enter your VBA function in this new window.

Accessing the OmniDriver COM Object from VBA

The following procedure assumes you are in the VBA project/module editing window.

► ***Procedure***

1. Click on Tools.References.
2. Click the **Browse** button.
3. Navigate to C:\Program Files\OceanOptics\OmniDriver\OOI_HOME.
4. Highlight OmniDriver32.dll and click the **Open** button.
5. Make sure **OmniDriver 2.0 Type Library** is checked and then click **OK**.

Passing Arguments into Your VBA Function

When you declare your function by typing it in the Fx box, you can specify:

- Hardcoded arguments, or
- Cell locations (e.g., A1, A2, B7, etc.)

Test Running Your VBA Function

► ***Procedure***

1. In your Excel worksheet, click on a cell and type **=YourFunctionName()** in the **Fx** box near the top of the window.
2. Hit the **Enter** key while the cursor is positioned in the **Fx** box to invoke your function.

   You may need to adjust the security level for Excel to permit the operation of your VBA macro. To do this:

   a. Select the **Tools | Options** menu item.
   b. Click on the **Security** tab.
   c. Click the **Macro Security** button and choose a security level that will permit your macro to run.

Developing in Visual Basic

Using the .NET Interface to OmniDriver

PREFERRED: To use the .NET assembly interface to OmniDriver in your Visual Basic application, you must add a reference to your assembly as follows:

► ***Procedure***

1. Click on the Project menu item and choose Add Reference.
2. Click on the Browse tab.
3. Navigate to the OOI_HOME directory.
4. Highlight the NETOmniDriver.dll and/or NETSpam.dll and click OK.

Using the COM Interface to OmniDriver

If you are using the COM interface to OmniDriver, you must add a reference to the new OmniDriver COM object within your Visual Basic project after you install OmniDriver.

► ***Procedure***

To add a reference to the OmniDriver COM object within Microsoft Visual Studio 2005:

1. Click on the **Project** menu item and choose **Add Reference**.
2. In the **Add Reference** dialog box, select the **COM** tab.
3. Scroll down to "OmniDriver 2.0 Type Library", highlight it, and click OK. If you do not see "OmniDriver 2.0 Type Library" in the list of available COM objects, it means you did not successfully register the new OmniDriver32.dll file.

If you need to use objects and methods belonging to the **SPAM** module, you must also add a new reference to SPAM 2.0 Type Library using the same procedure listed above.

If you need to register the OmniDriver32.dll and/or SPAM32.dll files, cd to the OOI_HOME directory containing OmniDriver32.dll and type **regasm32/codebase/tlb NETOmniDriver.dll** and/or **regasm32/codebase/tlb NETSpam.dll**.

If you are using the C/C++ interface to OmniDriver, make sure that the new OOI_HOME directory is in the system PATH.

Developing with Microsoft Visual Basic 6.0

You may use either the COM interface or the “C” function call interface to OmniDriver when developing with VB6. The COM interface is much simpler to use and is the recommended way to access OmniDriver from VB6 applications.

When using the COM interface, remember to use the “Set” keyword when assigning objects to VB variables. The following is an example of correct syntax:

```
Set GPIOFeatureController = wrapper.getFeatureControllerGPIO(0)
```

If you forget the "Set" keyword, you will get the run-time error message **Object doesn't support this property or method**.

Before you can access OmniDriver COM objects from your VB6 application, you must add a reference to the OmniDriver COM object. To do this, use the following procedure:

► ***Procedure***

1. Click on the VB6 menu item **Project.References…**.
2. In the list of available references, scroll down to the **OmniDriver .NET interface** and/or **SPAM .NET interface** line and check its checkbox.
3. Click **OK**.

If you choose to use the C interface (not recommended), we have provided the file OmniDriver.bas, located in VisualBasic6_Declarations subdirectory to expose OmniDriver's C methods. You must add this file to your VB6 project using the **Add Modules** option. More information on accessing OmniDriver functionality from VB6 is located in the **vb6_c_SpectrometerAcquisition** sample application.

Note

We do not provide function prototype declarations for SPAM library functions. If you need to use SPAM functions, you must use the COM interface to OmniDriver.

Modifying Your COM Application to Run on 64-bit Windows using the .NET Interface to OmniDriver

► ***Procedure***

1. Remove the reference to the old OmniDriver object.
2. Add a reference to the new NETOmniDriver and/or NETSpam objects. To do this:
    a. In Solution Explorer pane, expand the References"node.
    b. Right-click on **OmniDriver** and choose **Remove**.
    c. Right-click on **References** and choose **Add Reference…**
    d. Click on the **Browse** tab.
    e. Navigate to the OOI_HOME directory (typically C:\Program Files\Ocean Optics\OmniDriverSPAM\OOI_HOME)
    f. Select **NETOmniDriver.dll** and click **OK**.
    g. Repeat Step f for NetSpam.dll if you are using any SPAM functions

If you fail to make the above change, you may see a build error like the following:

> "The referenced component 'OmniDriver' could not be found."

Remove any calls to Wrapper.CreateWrapper(). This is no longer necessary.

Replace the symbolic boolean contants “true” and “false” used as input values to OmniDriver methods with 1 and 0 respectively. Do the same when using “true” and “false” to test the return value of any OmniDriver methods.

Troubleshooting Your Application

Runtime Errors

Symptom: FileNotFoundException was unhandled

The specified module could not be found.

Possible Cause: The OOI_HOME directory is not in the system PATH

Test Application

A simple test application named SpectrumTest64.exe is included in the OOI_HOME folder. You can run this app to verify OmniDriver was installed correctly.

This app is the vs2005_cpp_cpp_SpectrumTest sample

Modifying Your C++ Project Settings for 64-bit Windows

The samples that come with OmniDriver are initially set to generate 32-bit Windows EXEs. The following shows you how to modify the project settings for the samples to generate 64-bit EXEs.

Create an x64 Platform Option

► ***Procedure***

1. Choose **Configuration Manager** from the **Solution Platforms** drop-down menu at the top of the Visual Studio GUI.
2. Choose **<New…>** from the **Active solution platform** drop-down menu.
3. Choose x64 from the Type or select the new platform drop-down menu.
4. Choose **Win32** from the **Copy settings from** drop-down menu.
5. Check Create new project platforms.
6. Make sure the **x64** platform is selected when you build your application.

Change Your include/header File Location

If you installed the new 64-bit OmniDriver product to a new location, you must modify the project property settings of your Visual Studio project to specify the location of the 64-bit version of OmniDriver:

Project | Properties | C/C++ | General | Additional Include Directories

You may also need to change the setting for the location of the Java header files to find them where you installed the 64-bit Java JDK (typically C:\Program Files\Java\jdk1.6.0_21\include).

Change Your Library Dependencies

- Change common32.lib to common64.lib
- Change OmniDriver32.lib to OmniDriver64.lib
- Change SPAM32.lib to SPAM64.lib

You can change the library dependencies by navigating the menu hierarchy as follows:

Project | Properties | Linker | Input | Additional Dependencies

If you fail to make the above changes so that your application links to the 64-bit versions of the OmniDriver libraries, you will get a number of linker errors like the following:

“error LNK2001: unresolved external symbol Wrapper_Create”

You may also need to change the path to search for the above libraries. You can change this path by navigating the menu hierarchy as follows:

Project | Properties | Linker | General | Additional Library Directories

Output EXE Location

The EXE produced when you build your 64-bit application will be placed in a new location. By default, it will be located in <project folder>\x64\release (or <project folder>\x64\debug).

Chapter 10

Sample Programs

Overview

This section contains OmniDriver sample programs for the following programming languages:

- ***C***
- ***C++***
- ***C#***
- ***Delphi***
- ***Java***
- ***LabVIEW***
- ***Visual Basic***
- ***Other Samples***

C Samples

NationalInstrumentsCVI_Cinterface_Demo

Location: OmniDriver\samples\c\NationalInstrumentsCVI_Cinterface_Demo
Language: C, National Instruments CVI C compiler
Access method: uses the C interface to access OmniDriver functionality.
API: Wrapper
GUI: No
Demonstrates how to do the the following:
- Access a spectrometer
- Display spectrometer name and serial number
- Display wavelength coefficients
- Set acquisition parameters (integration time, etc.)
- Acquire a spectrum.

Borland_C_interface_sample

Location: OmniDrivr\win32\c
Language: C++
Access method: “C” interface

C++ Samples

vs2005_cpp_cpp_MultithreadedTimerTest

Language: C++, Microsoft Visual Studio 2005
Access method: uses the C++ interface to access OmniDriver functionality.
API: Wrapper
GUI: No
Demonstrates how to do the following:

- Control multiple spectrometers from multiple threads running concurrently
- Use `QueryPerformanceCounter()` to measure elapsed times

vs2005_cpp_cpp_SpectrumTest

Language: C++, Microsoft Visual Studio 2005
Access method: Uses the C++ interface to access OmniDriver functionality.
API: Wrapper
GUI: No
Demonstrates how to do the following (minimal application):

- Access a spectrometer
- Set acquisition parameters (integration time, etc.)
- Acquire a spectrum.

vs2005_cpp_c_SpectrometerTest

Language: C++, Microsoft Visual Studio 2005
Access method: Uses the C interface to access OmniDriver functionality.
API: Wrapper
GUI: No
Demonstrates how to do the following (minimal application):

- Access a spectrometer
- Display spectrometer name and serial number
- Display wavelength coefficients
- Set acquisition parameters (integration time, etc.)
- Acquire a spectrum.

vs2005_cpp_cpp_StrobeEnableSample

Language: C++, Microsoft Visual Studio 2005
Access method: Uses the C++ interface to access OmniDriver functionality.
API: Wrapper
GUI: No
Demonstrates how to do the following:

- Turn the lamp on and off
- Acquire spectra

vs2005_cpp_c_FeatureSample

Language: C++, Microsoft Visual Studio 2005

Access method: Uses the C interface to access OmniDriver functionality.
API: Wrapper
GUI: No
Demonstrates how to do the following:
- Use the GPIO (general purpose I/O) feature
- Use the continuous strobe feature
- Get API version
- Get API build number (of the Wrapper API)
- Get serial number, firmware version, spectrometer name
- Get and display wavelength coefficients
- Set acquisition parameters
- Get a spectrum and display pixel values

vs2005_cpp_cpp_TECSample

Language: C++, Microsoft Visual Studio 2005
Access method: uses the C++ interface to access OmniDriver functionality.
API: Wrapper
GUI: No
Demonstrates how to do the following:
- Use the thermoelectric cooling (TEC) feature

vs2005_cpp_cpp_TriggerDemoOne

Language: C++, Microsoft Visual Studio 2005
Access method: uses the C++ interface to access OmniDriver functionality.
API: Wrapper
GUI: No
Demonstrates how to do the following:
- Use external trigger mode 1 (software trigger mode)

vs2005_cpp_cpp_SingleShotSample

Language: C++, Microsoft Visual Studio 2005
Access method: Uses the C++ interface to access OmniDriver functionality.
API: Wrapper
GUI: No
Demonstrates how to do the following:
- Use external trigger mode 4 (single-shot mode, aka quasi-realtime acquisition mode)

vs2005_cpp_cpp_mfc_Odlumens2

Language: C++, Microsoft Visual Studio 2005
Access method: uses the C++ interface to access OmniDriver functionality.
API: Wrapper
GUI: Yes (MFC)
Demonstrates how to do the following:
- Perform CIE color calculations using functions from the SPAM library

vs2005_cpp_cpp_DLLAccess

Language: C++, Microsoft Visual Studio 2005
Demonstrates how to do the following:
- Access OmniDriver functionality from a user-written DLL (as opposed to calling OmniDriver functions from an EXE).

`vs6_cpp_cpp_MultiChannelDemo`

Language: C++, Microsoft Visual C++ 6.0
Access method: Uses the C++ interface to access OmniDriver functionality.
API: Wrapper
GUI: No
Demonstrates how to do the following:
- Access OmniDriver functionality in applications written using Microsoft Visual C++ 6.0
- Acquire a spectrum from a multi-channel spectrometer

`vs6_cpp_cpp_SpectrumTest`

Language: C++, Microsoft Visual C++ 6.0
Access method: Uses the C++ interface to access OmniDriver functionality.
API: Wrapper
GUI: No
Demonstrates the following:
- Minimalist implementation of the logic necessary to acquire a spectrum

Language: C++
IDE: Borland C++ Builder 2006
Access method: C
API: Wrapper
GUI: yes
Demonstrates the following:
- get the serial number of the spectrometer
- acquire a spectrum

C# Samples

`CSharpSpectrometerTest`

Language: C#
Access method: NET
API: Wrapper
GUI: No
Demonstrates the following:
- Absolute simplest demonstration of how to acquire a spectrum from a spectrometer

`CSharp_StrobeEnableSample`

Language: C#
Access method: NET

API: Wrapper
GUI: No
Demonstrates to do the following:
- Turn the lamp on and off.
- Use the Auto Strobe feature.

`CSharp_TECSample`

Language: C#
Access method: NET
API: Wrapper
GUI: No
Demonstrates the following:

-The Jaz Indy feature

- The External Temperature Wrapper feature (USB2000, USB4000). The temperature probe plugs into the USB-LS450, which then plus into the spectrometer.

- How to use the Thermo-electric Cooling (TEC) feature. This feature is available on several spectrometers, such as the QE65000, NIR256, and NIR512

- Pixel binning with the STS spectrometer

`ConcurrentAcquisitionThreads`

Language: C#
Access method: Uses the C interface to access OmniDriver functionality.
API: Wrapper
GUI: No
Demonstrates how to do the following:
- Acquire spectra from two spectrometers running concurrently on two separate threads.
  See also: Visual Basic sample which uses the COM interface to OmniDriver to drive two spectrometers running concurrently on two threads

`CSharp_SingleShotSample`

Language: C#
Access method: NET
API: Wrapper
GUI: No
Demonstrates how to do the following:
- Use Single Shot mode (Hardware Trigger mode 4) to set a relatively lengthy integration time but still control the timing of the start of acquisition fairly accurately (within 1/2 of the minimum integration time of your spectrometer) based on when you call `wrapper.getSpectrum()`.

`vs2005_net_PlotSpectrum`

Language: C#
Access method: NET
API: Wrapper
GUI: Yes
Demonstrates how to do the following:
- Acquire spectra and display them graphically

STSFeatures

IDE : Visual Studio 2010
Language: C#
Access method: NET
API: Wrapper
GUI: Yes
Demonstrates how to do the following:
- Pixel binning
- How to recover from a non-existent hardware trigger signal

Delphi Samples

Delphi_Borland_COM_Interface_sample

Language: Delphi
IDE: Borland Delphi for Microsoft Win32 (Borland Developer Studio 2006)
Access method: COM
API: Wrapper
GUI: Yes
Demonstrates how to do the following:
- Access a spectrometer
- Display spectrometer name and serial number
- Set acquisition parameters (integration time, etc.)
- Acquire a spectrum.
- Use the Thermo-electric Cooling (TEC) feature

Java Samples

ConcurrentAcquisitionThreads

Location: OmniDriver\samples\java\ConcurrentAcquisitionThreads
Language: Java
Access method: accesses OmniDriver functionality directly from the OmniDriver.jar file
API: Wrapper
GUI: Yes (NetBeans framework)
Demonstrates how to do the following:
- Access a spectrometer
- Display serial number
- Acquire spectra from two spectrometers concurrently using two threads
- Compute the average speed of acquisition, in spectra per second, for each spectrometer

Notes:
This sample should be run from within the NetBeans IDE as it requires certain jar files provided by the NetBeans application framework.

HighSpeedBasicSample

Location: OmniDriver\samples\java\HighSpeedBasicSample
Language: Java
Access method: accesses OmniDriver functionality directly from the OmniDriver.jar file
API: Wrapper
GUI: No (console app)
Demonstrates the following:
- How to use the high speed acquisition capabilities of the Wrapper API

Notes:

This app was developed using the NetBeans IDE. However, because of the large memory requirement of this sample, please run this application from the command-prompt using the run.bat supplied in the main folder for this sample
C:\Program Files\OceanOptics\OmniDriver\samples\java\HighSpeedBasicSample
Depending on the speed of your PC, this may take up to one minute to complete.

HighSpeedAcquisitionSample

Location: OmniDriver\samples\java\HighSpeedAcquisitionSample
Language: Java
Access method: Accesses OmniDriver functionality directly from the OmniDriver.jar file
API: Wrapper
GUI: No (console app)
Demonstrates how to do the following:
- Use the hardware trigger modes
- Use the high speed acquisition capabilities of the Wrapper API

Notes:

This application was developed using the NetBeans IDE. You can build it and run it within the IDE, or you can run it from the command-prompt using run.bat. To run this sample application from the command-prompt, use the "run.bat" command found in the samples/java/HighSpeedAcquisitionSample directory.

JavaWrapperSample

Location: OmniDriver\samples\java\JavaWrapperSample
Language: Java
Access method: Accesses OmniDriver functionality directly from the OmniDriver.jar file
API: Wrapper
GUI: No
Demonstrates how to do the following:
- Access a spectrometer
- Display serial number and firmware version number
- Use the Board Temperature feature
- Use the Continuous Strobe feature
- Use the LS450 feature
- Set acquisition parameters (integration time, boxcar width, etc.)
- Acquire a spectrum

Notes:

This sample application was developed using the NetBeans IDE. You can build it and run it within the IDE, or run it from the command-prompt. To do so, use the run.bat command found in the samples/java/JavaWrapperSample directory.

`MultiChannelSample`

Location: OmniDriver\samples\java\MultiChannelSample
Language: Java
Access method: accesses OmniDriver functionality directly from the OmniDriver.jar file
API: Wrapper
GUI: No (console app)
Demonstrates how to do the following:
- Access a multi-channel spectrometer
- Acquire spectra from two channels

Notes:

This sample application was developed using the NetBeans IDE. You can build it and run it within the IDE, or you can run it from the command-prompt.To do so, use the "run.bat" command found in the samples/java/MultiChannelSample directory.

`Simple`

Location: OmniDriver\samples\java\Simple
Language: Java
Access method: Accesses OmniDriver functionality directly from the OmniDriver.jar file
API: Wrapper
GUI: No (console app)
Demonstrates the following:
- Absolute minimum effort to acquire spectra

Notes:

This sample application was NOT developed using the NetBeans IDE. To build it, use build.bat.
To run it, use runtest.bat.

`SpectrometerTest`

Location: OmniDriver\samples\java\SpectrometerTest
Language: Java
Access method: accesses OmniDriver functionality directly from the OmniDriver.jar file
API: Wrapper
GUI: No (console app)
Demonstrates the following:
- Absolute minimum effort to acquire spectra

Notes:

This app was developed using the NetBeans IDE. You can build it and run it within the IDE, or you can run it from the command-prompt using run.bat

`ThreadingSpectrometer`

Location: OmniDriver\samples\java\ThreadingSpectrometer
Language: Java
Access method: Accesses OmniDriver functionality directly from the OmniDriver.jar file
API: Wrapper
GUI: No (console app)

Demonstrates how to do the following:
- Use hardware trigger modes
- Control a spectrometer from a background thread

Notes:

This sample application was developed using the NetBeans IDE. You can build it and run it within the IDE, or you can run it from the command-prompt using run.bat. After starting this application, you must generate a hardware trigger signal to cause it to acquire a spectrum.

`WavelengthCalibrationDemo`

Location: OmniDriver\samples\java\WavelengthCalibrationDemo
Language: Java
Access method: Accesses OmniDriver functionality directly from the OmniDriver.jar file
API: Wrapper
GUI: No (console app)
Demonstrates the following:

- How to set wavelength calibration coefficients into the spectrometer's EEPROM or into a temporary buffer

Notes:

This sample application was developed using the NetBeans IDE.
You can build it and run it within the IDE, or you can run it from the command-prompt using run.bat

LabVIEW Samples

When you start the LabVIEW samples you must inform the project of the location of your OmniDriver32.dll, common32.dll and SPAM32.dll files and the libraries provided by OmniDriver (e.g., Wrapper.llb). For each file click on **Browse...** and select the files or library function.

`ContinuousStrobeSample.vi`

`SetElectricDarkSample.vi`

`SpectrometerTestGPIO.vi`

`StrobeEnableSample.vi`

`SpectrometerTest.vi`

Language: LabVIEW 7.1 and 8.5
Access method: LabVIEW libraries provided by OmniDriver
API: Wrapper
GUI: Yes
Demonstrates how to do the following:
- Access a spectrometer
- Set acquisition parameters (integration time, etc.)
- Acquire a spectrum
- Display the spectrum in a simple graph

`TECSample.vi`

Language: LabVIEW 7.1 and 8.5
Access method: LabVIEW libraries provided by OmniDriver
API: Wrapper
GUI: yes
Demonstrates the following:
- How to use the Thermo-electric Cooling (TEC) feature supported by several spectrometers

Visual Basic Samples

`vb6_c_SpectrometerAcquisition`

Language: Visual Basic 6.0 (aka VB6)
Access method: Uses the C interface to access OmniDriver functionality.
API: Wrapper
GUI: Yes
Demonstrates how to do the following:
- Access a spectrometer
- Obtain the spectrometer name, serial number, and firmware version
- Use the Continuous Strobe feature
- Use the Thermo-electric Cooling (TEC) feature
- Use the GPIO feature (general purpose I/O)
- Determine number of pixels and number of dark pixels provided by spectrometer
- Set acquisition parameters (integration time, boxcar width, electric dark correction)
- Acquire a spectrum

`vs2005_net_ContinuousStrobe_and_SingleStrobe`

Language: Visual Basic
Access method: NET
API: Wrapper
GUI: yes
Demonstrates the following:
- How to use the Single-strobe and Continuous-strobe features

`vs2005_net_LS450Sample`

Language: Visual Basic
Access method: NET
API: Wrapper
GUI: yes
Demonstrates how to do the following:
- Access a spectrometer
- Obtain the spectrometer serial number
- Strobe enable (turn the lamp on/off)
- Set LS450 into pulsed/non-pulse mode

`vs2005_net_MultiThreading`

Language: Visual Basic
Access method: NET
API: Wrapper
GUI: yes
Demonstrates the following:
- How to acquire spectra from two spectrometers running simultaneously on two different threads

`vs2005_net_SPAMColorLAB`

Language: Visual Basic
Access method: NET
API: Wrapper
GUI: yes
Demonstrates how to do the following:
- Use the functions in the SPAM library to compute L*A*B* colorspace values
- Choose an observer and an illuminant
- Compute the X,Y,Z tristimulus values

`vs2005_net_WrapperDemo`

Language: Visual Basic
Access method: NET
API: Wrapper
GUI: Yes
Demonstrates how to do the following:
- Access a spectrometer
- Obtain the spectrometer serial number
- Strobe enable (turning the lamp on/off)
- GPIO (general purpose I/O bits)
- Thermo-electric temperature control
    - Set acquisition parameters (integration time, boxcar width, scans to average, electric dark correction)
- Acquire a spectrum

SPAM Samples

The following are samples for SPAM for C++, C# and Visual Basic:

C++: spam\win32\cpp\vs2005_cpp_cpp_Colortemperature

C++: spam\win32\cpp\vs2005_cpp_cpp_mfc_Odlumens2

C#: spam\win32\csharp\vs2005_SPAM_Peaks

Visual Basic: spam\win32\visualbasic\vs2005_SPAMColorLAB

Other Samples

Language: Excel, MatLab
Access method: COM
API: Wrapper
GUI: Yes
Demonstrates how to do the following:
- Access a spectrometer
- Display spectrometer name and serial number
- Set acquisition parameters (integration time, etc.)
- Acquire a spectrum

Chapter 11

Redistributing OmniDriver with Your Application

Overview

Customers who have purchased the OmniDriver product have a royalty-free right to redistribute the runtime version of OmniDriver along with the applications they have developed that depend upon OmniDriver to operate. Redistribution of OmniDriver components is subject to the terms and conditions of Ocean Optics license agreements.

We provide automated installers to facilitate redistributing the necessary components of OmniDriver to end-user computers so that, in most cases, it is not necessary to manually copy files to them. These new installers are located on the Ocean Optics FTP server at **ftp://downloads.oceanoptics.com/OmniDriver**. The name of each installer indicates the product, version, and target operating system. For example, use the file named OmniDriver_1.6_Win64_redistributable_installer.exe to install the OmniDriverSPAM on a 64-bit Windows PC. No password is required.

If you choose to install OmniDriver on your end-user's computer without requiring them to run a separate OmniDriver installation, use the following procedure.

► ***Procedure***

1. Copy the following directories (and everything beneath them) to your customer's PC:
   - OOI_HOME
   - _jvm (this directory *must* be placed on a peer level with the OOI_HOME directory)
   - ezusb_driver (32-bit Windows platform only)
   - winusb_driver (64-bit Windows platform only)
   - labview (optional: if your application was written in LabVIEW)
2. Define the OOI_HOME environment variable. Give it a value like:
   C:\Program Files\Ocean Optics\OmniDriver\OOI_HOME
   Essentially, set this variable to the location of the OOI_HOME directory.
3. Add the pathname of the OOI_HOME directory to your systems PATH environment variable.
4. On 64-bit Windows systems: cd into the winusb_driver folder and issue the command
   dpinst64.exe q /lm /sa
   This installs the signed WinUSB driver files into the Windows system area. As a result, when you plug in your spectrometer, it will automatically find and install the required driver.

5. Optional: If your application relies on COM (deprecated now that we are supporting .NET assemblies), you must issue the following command(s) to expose the OmniDriver and/or SPAM assemblies as COM objects:
   RegAsm32.exe /codebase /tlb NETOmniDriver.dll
   RegAsm32.exe /codebase /tlb NETSpam.dll (if your app uses the SPAM functions)
   The RegAsm32.exe utility is provided in the OOI_HOME directory. Also included is RegAsm64.exe if you are targeting a 64-bit Windows platform.

Deploying Your C# Application

Normally, all that is needed to deploy your C# application is the EXE file containing the application itself. And you will also need to deploy the appropriate OmniDriver "redistributable" installer (no password required by this installer).

However, if your C# application uses the OmniDriver .NET assembly interface, you must also ensure that a copy of NETOmniDriver.dll and/or NETSpam.dll is placed in the same folder as your application's EXE file. You can obtain these two DLL files from the OOI_HOME directory.

For Mac Users

It is not necessary to install Java because it is preinstalled on the Mac. The Mac comes with USB support built-in. So there is no need to download or install a USB driver.

► *Procedure*

1. Define the OOI_HOME environment variable to point to the location of the OOI_HOME directory.
2. Add the OOI_HOME directory to the system PATH environment variable.

For Linux Users

► *Procedure*

1. Make sure that the latest version of the open-source Linux USB driver named "libusb" has been installed.
2. Copy the file **10-oceanoptics.rules** from the **Drivers** directory into /etc/rules.d and then restart udev. On Fedora and similar Linux distros, you can restart udev with the following commands:
   - udevcontrol reload_rules
   - udevstart

   If this does not successfully restart udev, you will need to reboot.
3. Define the OOI_HOME environment variable to point to the location of the OOI_HOME directory.
4. Add the OOI_HOME directory to the system PATH environment variable.

Chapter 12

Wrapper API Reference

Overview

The Wrapper API is the collection of objects and methods your application uses to control spectrometers and acquire data from them. This section gives detailed information about a few of these functions. For a complete description of all objects and methods comprising the “Wrapper API”, refer to the javadocs for the OmniDriver and SPAM libraries.

getCalibrationCoefficientsFromEEProm()

Read out all calibration coefficents from the spectrometer’s EEPROM. This includes coefficients for stray light correction, wavelength calibration, and non-linearity calibration.

Visual Basic – COM interface

```
Dim coefficients As OmniDriver.CCoCoefficients
coefficients =
wrapper.getCalibrationCoefficientsFromEEProm(spectrometerIndex)
listBox.Items.Add("getStrayLightSlope = " &
coefficients.getStrayLightSlope())
listBox.Items.Add("getStrayLight = " & coefficients.getStrayLight())
listBox.Items.Add("getWlIntercept = " & coefficients.getWlIntercept())
listBox.Items.Add("getWlFirst = " & coefficients.getWlFirst())
listBox.Items.Add("getWlSecond = " & coefficients.getWlSecond())
listBox.Items.Add("getWlThird = " & coefficients.getWlThird())
```

highSpdAcq_AllocateBuffer()

This is the first highSpdAcq method you should call when using high-speed data acquisition mode. This method will allocate internal buffer space to accommodate the specified number of spectra.

Parameters

```
void highSpdAcq_AllocateBuffer(spectrometerIndex,numberOfSpectraToAcquire)
```

`int spectrometerIndex` 0-N, specifies which attached spectrometer to access
`int numberOfSpectraToAcquire` should be in the range 1-3000

highSpdAcq_GetNumberOfSpectraAcquired()

Call this method to reliably determine the number of spectra acquired by the preceding call to `highSpdAcq_StartAcquisition()`. In most cases it will return the same value you specified for the `numberOfSpectraToAcquire` parameter of the `highSpdAcq_AllocateBuffer()` method. However, if you called `highSpdAxq_StopAcquisition()` before all requested spectra have been collected, the number returned by this method will be smaller.

Parameters

```
int highSpdAcq_GetNumberOfSpectraAcquired()
```

returns int the number of spectra successfully acquired.

highSpdAcq_GetSpectrum()

Call this method to obtain the actual spectra acquired in high-speed data mode. This method should be called *after* you have called `highSpdAcq_StartAcquisition()`. Normally you will place calls to this method inside a loop, once for each spectrum acquired.

Parameters

```
double[] highSpdAcq_GetSpectrum(spectrumNumber)
```

int spectrumNumber 0-N, specifies which spectrum from the internal buffer area to return

returns double[] an array of pixel values constituting the spectrum. Returns null if no spectra have been acquired yet, or if fewer spectra have been acquired than the spectrumNumber you specified.

highSpdAcq_GetTimeStamp()

This method returns a time stamp associated with the specified spectrum. In high-speed data acquisition mode, many spectra are acquired and stored internally by means of a single call to the `highSpdAcq_StartAcquisition() method`. OmniDriver generates a time stamp at the moment each spectrum is acquired and stores it along with the spectrum itself. Thus, your application can precisely identify the time at which each individual spectrum was acquired.

Parameters

```
HighResTimeStamp highSpdAcq_GetTimeStamp(spectrumNumber)
```

int spectrumNumber 0-N, specifies which spectrum from the internal buffer area to return

returns a `HighResTimeStamp` object which specifies the time of acquisition for the corresponding spectrum. Returns null if no spectra have been acquired yet, or if fewer spectra have been acquired than the `spectrumNumber` you specified.

highSpdAcq_IsSaturated()

Tells you whether the specified spectrum, acquired via high-speed data acquisition mode, was saturated.

Parameters

```
boolean highSpdAcq_IsSaturated(spectrumNumber)
```

int spectrumNumber 0-N, specifies the spectrum for which we want to know if it was saturated

returns boolean true if the spectrum was saturated, or if no spectra have been acquired yet, or if fewer spectra have been acquired than the `spectrumNumber` you specified. Returns false if the spectrum exists and was not saturated.

highSpdAcq_StartAcquisition()

This method initiates collection of spectra in high-speed data acquisition mode. It will not return until either the specified number of spectra have been acquired, or you call `highSpdAcq_StopAcquisition()`. You should first call the `highSpdAcq_AllocateBuffer()` to specify the number of spectra you wish to acquire.

Parameters

```
void highSpdAcq_StartAcquisition(spectrometerIndex)
```

int spectrometerIndex 0-N, specifies which attached spectrometer to access

highSpdAcq_StopAcquisition()

This method aborts the currently active high-speed acquisition process. When you call this method, it will cause the `highSpdAcq_StartAcquisition()` method to return as soon as the current spectrum has been acquired, even if this results in fewer spectra being acquired than you originally specified when you called the `highSpdAcq_AllocateBuffer()` method.

This method must be called from a *different* thread than the thread from which the `highSpdAcq_StartAcquisition()` method was called because `highSpdAcq_StartAcquisition()` will block (i.e., not return to the caller) until all requested spectra have been acquired.

It is normally not necessary to call this method. Call this method only if you wish to interrupt the acquisition process prematurely.

Parameters

```
void highSpdAcq_StopAcquisition()
```

No parameters or return values.

setBoxcarWidth()

Calling Conventions:

Java

```
   wrapper.setBoxcarWidth(int spectrometer, int
numberOfPixelsOnEitherSideOfCenter)
```

C

```
   Wrapper_setBoxcarWidth()
```

C++

LabVIEW

VisualBasic

setCalibrationCoefficientsIntoBuffer()

This method writes the specified coeffients into a buffer area used by OmniDriver to correct raw pixel values when returning spectra to the user. This method does *not* write the coefficients into the EEPROM of the spectrometer. To protect against accidental destruction of the calibration coefficients in the spectrometer, you must first call the `insertKey("Mat429sky")` method before this method will do anything.

Use this method (rather than the EEPROM flavor) when you want to test the effect of new calibration coefficients, but do not want to write them into the spectrometer itself.

Refer to the description of the setCalibrationCoefficientsIntoEEProm() method below for more details.

setCalibrationCoefficientsIntoEEProm()

This method writes various calibration coefficients into the EEPROM of the spectrometer. If invalid calibration coefficients are written into the EEPROM, the spectrometer will no longer return valid data, so be very careful with the use of this method. To protect against accidental destruction of the calibration coefficients in the spectrometer, you must first call the `insertKey("Mat429sky")` method before this method will do anything.

Recommended: If you want to test the effect of your new calibration settings before permanently burning them into the spectrometer's EEPROM, use the `setCalibrationCoefficientsIntoBuffer()` method. This method causes all subsequent calls to `getSpectrum()` to use your new settings, but does not write them into the EEPROM of the spectrometer. This allows you to see the effect of your new coefficients.

Parameters

```
boolean setCalibrationCoefficeientsIntoEEProm(spectrometerIndex,
coefficients, applyWavelengthCoefficients, applyStrayLightConstant,
applyNonlinearityCoefficients)
```

int spectrometerIndex 0-N, specifies which attached spectrometer to access

Coefficients coefficients a data structure you must initialize with settings for any of the calibration coefficients you wish to set in the spectrometer EEPROM area

boolean applyWavelengthCoefficients must be true for wavelength coefficient settings to be written into EEPROM, otherwise these values will be ignored

boolean applyStrayLightConstant must be true for the stray light constant to be written into the EEPROM

boolean applyNonlinearityCoefficients must be true for the nonlinearity coefficients to be written into EEPROM

See also:
  setCalibrationCoefficientsIntoBuffer()

Visual Basic

```
Dim coefficients As New OmniDriver.CCoCoefficients
Dim spectrometerIndex As Integer

coefficients.CreateCoefficients()

coefficients.setWlIntercept(177.7322)
coefficients.setWlFirst(0.380633)
coefficients.setWlSecond(-0.00001346845)
coefficients.setWlThird(-2.9798580E-009)
coefficients.setStrayLight(3.0) ' slope is assumed to be 0
'alternatively: coefficients.setStrayLightB(1.1, 2.0) ' intercept,slope
coefficients.setNlCoef0(2.0) ' bogus numbers for demonstration only
coefficients.setNlCoef1(2.1)
coefficients.setNlCoef2(2.3)
coefficients.setNlCoef3(2.4)
coefficients.setNlCoef4(2.5)
coefficients.setNlCoef5(2.6)
coefficients.setNlCoef6(2.7)
coefficients.setNlCoef7(2.8)
coefficients.setNlOrder(7)

wrapper.insertKey("Mat429sky") ' enable setCalibrationCoefficients()
spectrometerIndex = 0
' In the following call, the three boolean arguements are as follows:
' boolean applyWavelengthCoefficients, boolean applyStrayLightConstant,
' and boolean applyNonlinearityCoefficients)
wrapper.setCalibrationCoefficientsIntoEEProm(spectrometerIndex, coefficients,
True, True, True)
```

setIntegrationTime()

Units: microseconds.

See *Why doesn't getSpectrum() return when I think it should?* In Appendix A for more information.

Calling Conventions

C interface

```
Dim wrapper_handle as int32
Dim spectrometer_index as int32
Dim integration_time as int32 ‘ units are in microseconds
Wrapper_setIntegrationTimeTime(wrapper_handle, spectrometer_index,
integration_time)
```

Chapter 13

SPAM Library

Overview

Ocean Optics provides browser-based javadocs documentation for the individual SPAM objects and methods. You can find the SPAM javadocs in the “docs\javadocs.spam” subdirectory of your OmniDriver installation directory. Double-click on the file index.html in this directory. This will open the SPAM documentation in your browser.

Perhaps the best source of information on how to use the SPAM library is provided by the sample applications that are provided with the OmniDriverSPAM and SPAM software products.

All OmniDriver and SPAM samples are combined into a single installer, named Windows_OmniDriver_and_SPAM_Sample_Pack. You can download the installer from **http://www.oceanoptics.com/Technical/softwaredownloads.asp**. No password is required to install the samples.

By default, all samples are installed into directories beneath a common root directory: C:\Ocean Optics\OmniDriverSamplePack-1.65 (1.65 is the version number, and it will change).

The following SPAM samples are provided for the Windows operating system:

Language	**Name of Sample**	**Location**
C#	vs2005_SPAM_Peaks	samples\spam\win32\csharp\vs2005_SPAM_Peaks
VB	vs2005_SPAMColorLAB	samples\spam\win32\visualbasic\vs2005_SPAMColorLAB
C++	ColorTemperature	samples\spam\win32\cpp\vs2005_cpp_cpp_ColorTemperature
C++	SpectrumPeaks	samples\spam\win32\cpp\vs2005_cpp_cpp_SpectrumPeaks
C++	ODLumens2	samples\spam\win32\cpp\vs2005_cpp_cpp_mfc_Odlumens2
C	AdvancedColor	samples\spam\win32\cpp\vs2005_cpp_c_AdvancedColor

Notes

Although these samples are located beneath a directory named win32, you may easily change the Visual Studio settings to build them in 64-bit mode.

All Windows samples were created using Microsoft Visual Studio 2005. However, if you are using a newer version of Visual Studio, you can easily use the Visual Studio conversion wizard to import these projects into your IDE.

Important Tips

- Always use the **SpectralMath** wrapper class to create any SPAM objects you need. This is necessary to avoid memory leaks in your application. The sample applications show how to do this.
- Remember to explicitly call the Dispose() method if you are writing a .NET application. Calling the Dispose() method on your SPAM objects when you no longer need those objects prevents memory leaks. This is only necessary if you are using the .NET interface to access OmniDriver and SPAM objects.
- If your app is multi-threaded, it is recommended that you create a separate **SpectralMath** instance for each thread.

C++ Sample

The following is an example of what a C++ SPAM application should look like. This is the source code from the C++ sample named “SpectrumPeaks”.

```
#include "stdafx.h"

#include "Wrapper.h"
#include "SpectralMath.h"
#include "AdvancedPeakFinding.h"

int _tmain(int argc, _TCHAR* argv[])
{
  double baseline;
  int indexOfPeak;
  int minimumIndicesBetweenPeaks;
  int numberOfSpectrometers;
  int peakIndex;
  int startingIndex;
  DoubleArray yDoubleArray;
  DoubleArray xDoubleArray;
  Wrapper* pWrapper;

  pWrapper = new Wrapper();
  numberOfSpectrometers = pWrapper->openAllSpectrometers();
  printf("numberOfSpectrometers = %d\n",numberOfSpectrometers);
  yDoubleArray = pWrapper->getSpectrum(0);
  xDoubleArray = pWrapper->getWavelengths(0);
```

```
   // Now use the SPAM functions to identify the peaks
   SpectralMath* pSpectralMath = new SpectralMath();
   peakIndex = 100;
   AdvancedPeakFinding advancedPeakFinding;
   advancedPeakFinding = pSpectralMath->createAdvancedPeakFindingObject();
       startingIndex = 0;
       minimumIndicesBetweenPeaks = 100;
       baseline = 100.0;
       do
       {
         indexOfPeak = advancedPeakFinding.getNextPeakIndex(yDoubleArray,
           startingIndex, minimumIndicesBetweenPeaks, baseline);
         if (indexOfPeak == 0)
              break;
         printf("index of peak = %d\n",indexOfPeak);
         startingIndex = indexOfPeak;
       } while (indexOfPeak > 0);

   printf("peak X value %f\n",spectrumPeak.getPeakXValue());
   delete pSpectralMath;

   return 0;
}
```

CSharp Sample

The following is an example of what a C# SPAM application should look like. This is the source code from the CSharp sample named "vs2005_SPAM_Peaks".

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

// Don't forget to add references to "NETOmniDriver" and "NETSpam"

namespace vs2005_SPAM_Peaks
{
    public partial class Form1 : Form
    {
        int numberOfSpectrometersFound;
        int spectrometerIndex; // indicates which spectrometer we are using
        OmniDriver.CCoWrapper wrapper;
        SPAM.CCoSpectralMath spectralMath;

        public Form1()
        {
            InitializeComponent();

            // Initialize the wrapper object
            wrapper = new OmniDriver.CCoWrapper();
            spectralMath = new SPAM.CCoSpectralMath();

            spectrometerIndex = -1; // set it to an invalid value
        }

        private void buttonDiscoverSpectrometers_Click(object sender,
                      EventArgs e)
        {
            numberOfSpectrometersFound = wrapper.openAllSpectrometers();
            if (numberOfSpectrometersFound < 1)
            {
                listBoxMessages.Items.Add("No spectrometers found");
                spectrometerIndex = -1; // set it to an invalid value
                return;
            }
            spectrometerIndex = 0; // arbitrarily choose the first
                 // spectrometer for this demo
            listBoxMessages.Items.Add("Selecting spectrometer: " +
            wrapper.getName(spectrometerIndex));
        }

        private void buttonFindPeaks_Click(object sender, EventArgs e)
        {
            double baseline;
            int indexOfPeak;
            int minimumIndicesBetweenPeaks;
            int numberOfPixels; // number of CCD elements/pixels provided by
                 // the spectrometer
            double[] spectrum;
            int startingIndex;

            if (spectrometerIndex == -1)
                return; // no available spectrometer
            numberOfPixels = wrapper.getNumberOfPixels(spectrometerIndex);

            // Set some acquisition parameters and then acquire a spectrum
            wrapper.setIntegrationTime(spectrometerIndex, 500000);//.5 second
            wrapper.setBoxcarWidth(spectrometerIndex, 10);
            wrapper.setCorrectForElectricalDark(spectrometerIndex, 1);
            spectrum = (double[])wrapper.getSpectrum(spectrometerIndex);

            // Display the raw pixel values of this spectrum
            //for (int index = 0; index < numberOfPixels; ++index)
            //{
            //    listBoxMessages.Items.Add("pixel[" + index + "] = " +
    //                 spectrum[index]);
            //}

            SPAM.CCoAdvancedPeakFinding advancedPeakFinding;
            advancedPeakFinding =
       spectralMath.createAdvancedPeakFindingObject();

            startingIndex = 0;
            minimumIndicesBetweenPeaks = 100;
            baseline = 100.0;
            do
            {
                indexOfPeak = advancedPeakFinding.getNextPeakIndex(spectrum,
          startingIndex, minimumIndicesBetweenPeaks, baseline);
                if (indexOfPeak == 0)
                    break;
                listBoxMessages.Items.Add("index of peak = " + indexOfPeak);
                startingIndex = indexOfPeak;
            } while (indexOfPeak > 0);

            advancedPeakFinding.Dispose(); // this is necessary to avoid
                  // memory leak
            listBoxMessages.Items.Add("all done");
        }

        private void buttonExit_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

Appendix A

FAQs

How fast can I acquire spectra?

The speed at which you can acquire spectra depends on the following factors:

- Minimum integration time
- Communication speed of the USB connection. Most PCs use USB 2.0, but some may still be using USB 1.0, which is much slower.
- Number of pixels of data. Some detectors return as few as 256 pixels of data, others may return up to 4096 pixels. Each pixel is transmitted as a 2-byte integer. The greater the number of pixels, the longer it takes to transmit the data.
- Speed of your PC.
- The number of spectrometers operating in parallel.

Consider each of these factors carefully when estimating the maximum rate at which you can acquire spectra on your PC. It is always best to perform actual real-world measurements.

The maximum possible spectral acquisition rate is achieved by use of the High-speed Data Acquisition mode. This is a specialized mode that imposes certain restrictions to boost speed. Refer to the **High Speed Data Acquisition** section for more information about this capability.

Why doesn't getSpectrum() return when I think it should?

When you set the integration time for a spectrometer, and then call `getSpectrum()`, you might expect that the duration of the call to `getSpectrum()` would match the integration time exactly. This is rarely the case, and there are several reasons why.

- When you first power-up the spectrometer by plugging it in to your PC, the spectrometer immediately begins acquiring spectra using default settings for integration time and other acquisition parameters. This is called Normal mode. As long as the spectrometer remains in Normal mode, and remains plugged in to your PC, it will continuously acquire spectra.

  When your application calls `getSpectrum()`, the spectrometer may be just beginning to acquire a new spectrum, or it may be nearly finished acquiring a spectrum, or it may be anywhere in between these two extremes. As a consequence, `getSpectrum()` may return almost immediately, or it may not return until the full integration period has elapsed; it just depends on how far along the spectrometer happened to be in its acquisition process at the moment you called `getSpectrum()`.

If your application calls `getSpectrum()` repeatedly (in a tight loop), the duration of subsequent calls to `getSpectrum()` will exactly match the integration time. This is because once you have called `getSpectrum()`, you are effectively syncronized with the operation of the spectrometer. Your first call to `getSpectrum()` will return at the exact moment that the spectrometer has completed a spectrum acquisition. Your second call to `getSpectrum()` occurs just as the spectrometer is beginning the next spectrum acquisition. Thus your second (and all subsequent) call to `getSpectrum()` must wait for the full integration period to elapse before the spectrometer has finished acquiring the next spectrum.

- There is a second reason why the duration of the `getSpectrum()` method may not match the integration time setting. Whenever you change one of the acquisition parameters that can affect the spectrum data (e.g., changing the integration time), OmniDriver will automatically take a stability scan. This means OmniDriver will ignore the spectrum that was being acquired at the moment when you called `getSpectrum()`. This spectrum was acquired using previous settings for the acquisition parameters, and thus is invalid. OmniDriver will return the *following* spectrum, which is based on the new acquisition parameter settings.

If I change spectrometer models, do I need to change my program?

No. As long as you stay within the Wrapper API, your software does not need to be modified when you change to a different type of Ocean Optics spectrometer.

However, keep in mind that certain features (e.g., "Thermo-electric Cooling") are only available on selected spectrometers. Refer to *Optional Features* for a list of features and which spectrometers support each of them.

Can my 32-bit OmniDriver application run on 64-bit Windows 7?

As of release 2.11, all of our 32-bit OmniDriver installers may be run on either 32-bit or 64-bit Windows. This allows you to create a single 32-bit application that can run on both 32-bit and 64-bit Windows systems.

Prior to release 2.11 of OmniDriver it was necessary to use the following manual procedure to install 32-bit OmniDriver on 64-bit Windows systems:

► ***Procedure***

1. Install 64-bit OmniDriver. This is necessary to obtain the winusb driver.

2. Uninstall 64-bit OmniDriver. This will leave the winusb driver intact.
   This step is optional, but it will help avoid confusion later on.
3. Install 32-bit OmniDriver.
4. In the C:\Program Files (x86)\Ocean Optics\OmniDriverSPAM\OOI_HOME directory:
   a. Rename NatUSB.dll → NatUSB_ezusb.dll
   b. Rename NatUSB_winusb.dll → NatUSB.dll
5. Build your app in 32-bit mode.
6. Run your application.

Can I use Visual Basic 6.0 (VB6) on 64-bit Windows 7?

Yes and no.

The VB6 IDE will only run on 32-bit editions of Windows (XP, Vista, or Win 7).

If you install the 32-bit version of OmniDriver on 64-bit Windows 7, your VB6 app should run just fine on 64-bit Windows 7. Your VB6 app may use either our "C" interface to OmniDriver or our "COM" interface to OmniDriver. Both will work on 64-bit Windows 7.

See ***http://msdn.microsoft.com/en-us/vbasic/ms788708*** for a very clear and comprehensive official statement of Microsoft's position regarding support for VB6 on XP, Vista, and Windows 7.

What happens if the timeout period is shorter than the integration time?

Some of the calls to getSpectrum() will timeout, and some will return valid spectra. The ratio of valid spectra to timeouts will depend on the ratio of your timeout period to the integration time. However, this situation should not cause you to miss spectral acquisitions, unless the integration time is so short your program cannot call getSpectrum() soon enough to capture the next spectrum.

What happens in the following scenario?

1. Set spectrometer to one of the external trigger modes
2. Call getSpectrum()
3. Timeout occurs before the trigger occurs
4. Trigger event happens before you call getSpectrum() a second time
5. Call getSpectrum()

The second call to getSpectrum() will return immediately with the spectrum that was acquired when the trigger event occurred.

If multiple trigger events happen after the first getSpectrum() has timed-out and before you call getSpectrum() a second time, when you do call getSpectrum() the second time, it immediately returns with the spectrum from the FIRST trigger event. Data from any additional trigger events during this window will be lost.

If a timeout occurs before the spectrum can be acquired, the spectral array returned by getSpectrum() will contain all zeroes.

By default, the timeout period is set to “infinity”. If you set a non-zero timeout and later wish to turn-off the timeout feature, you can do this by calling wrapper.setTimeout() with a value of zero. This causes getSpectrum() to wait “forever”.

Appendix B

Upgrading Your Application for OmniDriver 1.6

Overview

This appendix provides instructions for upgrading the following applications for OmniDriver 1.6:

- *C/C++ Applications Using the C++ Interface to OmniDriver*
- *C# Applications*
- *C++ Applications Using the COM Interface to OmniDriver*
- *Visual Basic COM Applications*
- *The Visual Basic 6.0 (VB6) IDE* only runs on 32-bit Windows platforms, and VB6 will only generate 32-bit executables. However, you can still run these 32-bit executables on 64-bit Windows 7 systems. To run your 32-bit VB6 application on a 64-bit Windows 7 system, you must install the 32-bit version of OmniDriver on that Windows 7 system. See Appendix A: FAQ "Can *my 32-bit OmniDriver application run on 64-bit Windows 7?"* for the correct procedure for installing 32-bit OmniDriver on 64-bit Windows 7 systems.
- VB6 Applications Using the COM Interface
- *VB6 Applications Using the "C" Interface*

C/C++ Applications Using the C++ Interface to OmniDriver

► *Procedure*

1. Define an environment variable named JAVA_HOME which points to the location where you installed the Java JDK. For example: C:\Program Files\Java\jdk1.6.0_21. You should see a directory named **include** directly beneath the JAVA_HOME directory.
2. You must restart your IDE after defining JAVA_HOME for this new environment variable to be accessible.
3. You can then modify the properties of your Visual Studio project to use this environment variable to specify the location of required header files. For example: $(JAVA_HOME)\include.

When you install OmniDriver 1.6, a new environment variable will be defined: OMNIDRIVER_HOME.

OMNIDRIVER_HOME will be set to the value C:\Program Files\Ocean Optics\OmniDriverSPAM (or wherever you have installed the OmniDriver product).You can then modify the properties of your Visual Studio project to use this environment variable to specify the location of required header files. Example: $(OMNIDRIVER_HOME)\include.

To build your C++ application, you must define following header file paths:

- $(OMNIDRIVER_HOME)\include
- $(JAVA_HOME)\include
- $(JAVA_HOME)\include\win32

You will also need the following library path:

- $(OMNIDRIVER_HOME)\OOI_HOME

You must also specify the following linker "Additional Dependencies"

- OmniDriver32.lib
- common32.lib
- optional: Spam32.lib

Note for 64-bit platform

If you are building your application to run on a 64-bit platform, change all library references to their 64-bit form.

- OmniDriver64.lib
- common64.lib
- Spam64.lib

Troubleshooting

Symptom: UnsatisfiedLinkError: NatUSB.dll (can't find dependent libraries)

Solution: Uninstall the old driver for your spectrometer(s). Then reinstall your spectrometer using the new WinUSB driver.

C# Applications

If you attempt to rebuild your 1.5 application after installing 1.6, you may get the error "The referenced component 'OmniDriver' could not be found." or "The type or namespace 'OmniDriver' could not be found…". Use the following procedure to prevent these errors.

► *Procedure*

To convert the OmniDriver 1.5 COM application to use the OmniDriver 1.6 .NET object/assembly, use the following procedure:

1. Expand the References node in the Solution Explorer for your project.

2. Right-click on **OmniDriver** and choose **Remove**.
3. Right-click on **References** and choose **Add Reference**.
4. Click on the **Browse** tab.
5. Navigate to the OOI_HOME directory. Highlight **NETOmniDriver.dll**, and click **OK**.
6. You may need to do the same for NETSpam.dll if your application uses functions from the SPAM library.
7. If you attempt to rebuild your application at this point, you will see errors like "The type of namespace name 'CCoWrapperClass' does not exist in the namespace 'OmniDriver (are you missing an assembly reference?)"
   To solve this... replace "CCoWrapperClass" with "CCoWrapper".
8. Remove calls to the wrapper.CreateWrapper() method. It is no longer necessary to call this method.

Note

It is still necessary to call the `Create`*`ObjectName`*`()` methods for SPAM objects which take one or more arguments. For example, you should continue to call the method `CIELAB.CreateCIELAB(myCIEColorObject)` because it takes an argument. But you do *not* need to call `AdvancedColor.CreateAdvancedColor()` because it does not take an argument.

---

9. When you rebuild your application, you may see one of the following error messages: "Operator '= =' cannot be applied to operands of type 'short' and 'bool', or “The best overloaded method match for SOME_METHOD() has some invalid arguments”.
   The solution is simply to replace the true/false symbolic values with 1 or 0, respectively.

C++ Applications Using the COM Interface to OmniDriver

You must add a reference to the new .NET assembly.

1. Click on menu item **Project | References…**
2. Expand Common Properties and click on References.
3. Click the **Add Path** button.
4. Browse to the OOI_HOME directory and add the **NETOmniDriver.dll**.

Visual Basic COM Applications

This section contains the procedure to convert Visual Basic COM Applications to use the OmniDriver 1.6 .NET object/assembly.

If you attempt to rebuild your 1.5 application after installing OmniDriver 1.6, you may get the error "Cannot load type library for reference 'OmniDriver'…"or "The referenced component 'OmniDriver' could not be found." Use the following procedure to prevent these errors.

► *Procedure*

1. Remove the reference to the old OmniDriver COM object (if necessary):
   a. In the Solution Explorer window, expand the References node. If the References node is not visible, try rebuilding your application (it will report the above-mentioned error). This will cause the References node to appear in the Solution Explorer window.
   If the References node does *not* appear, it means the IDE cannot find the old reference. In this case there is nothing that you need remove, so proceed to Step 2 below.
   b. Click on the "OmniDriver" reference and choose "Remove".
2. Click on the menu item **Project | Add Reference**.
3. Click on the **Browse** tab of the **Add Reference** dialog box.
4. Navigate to the OOI_HOME directory and highlight **NETOmniDriver.dll** (typically located in C:\Program Files\Ocean Optics\OmniDriver\OOI_HOME). Click **OK.**

To declare an instance of the new OmniDriver .NET assembly:

Dim wrapper As New OmniDriver.CCoWrapper

Remove all calls to wrapper.CreateWrapper(): This method is no longer needed. However, you should continue to call the "Create" methods for SPAM objects when these "Create" methods accept one or more parameters.

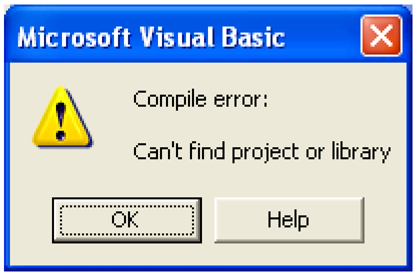
VB6 Applications

Caution: VB6 Limitation

The Visual Basic 6.0 (VB6) IDE only runs on 32-bit Windows platforms, and VB6 will only generate 32-bit executables. However, you can still run these 32-bit executables on 64-bit Windows 7 systems. To run your 32-bit VB6 application on a 64-bit Windows 7 system, you must install the 32-bit version of OmniDriver on that Windows 7 system. See Appendix A: FAQ "*Can my 32-bit OmniDriver application run on 64-bit Windows 7?*" for the correct procedure for installing 32-bit OmniDriver on 64-bit Windows 7 systems.

VB6 Applications Using the COM Interface

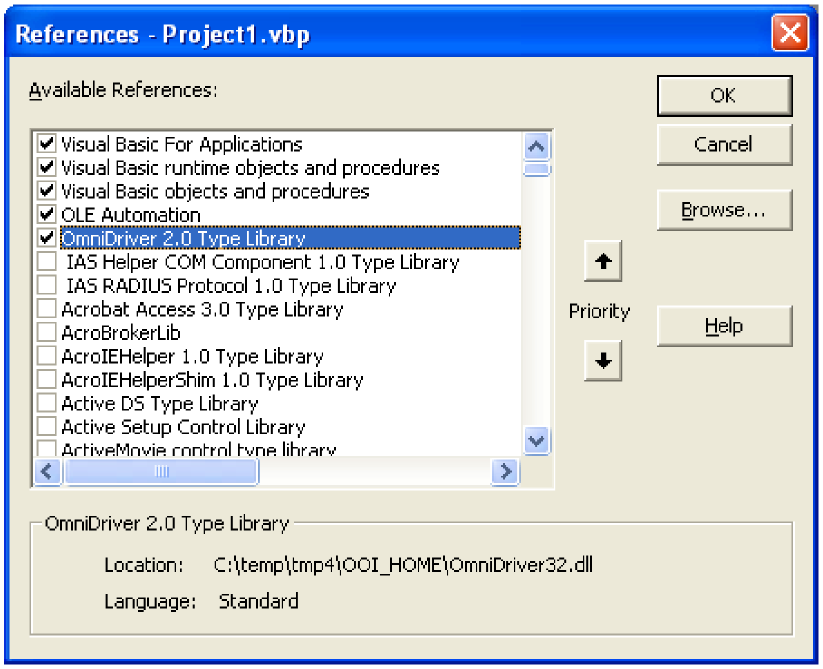
If you try to make an EXE without changing the reference to the OmniDriver COM object, you will see the following error message:


Microsoft Visual Basic

Compile error:

Can't find project or library

This error message is associated with your declaration(s) of OmniDriver COM objects. Click **OK** and then the References dialog box will appear:


References - Project1.vbp

Available References:

- [✓] Visual Basic For Applications
- [✓] Visual Basic runtime objects and procedures
- [✓] Visual Basic objects and procedures
- [✓] OLE Automation
- [✓] OmniDriver 2.0 Type Library
- [ ] IAS Helper COM Component 1.0 Type Library
- [ ] IAS RADIUS Protocol 1.0 Type Library
- [ ] Acrobat Access 3.0 Type Library
- [ ] AcroBrokerLib
- [ ] AcroIEHelper 1.0 Type Library
- [ ] AcroIEHelperShim 1.0 Type Library
- [ ] Active DS Type Library
- [ ] Active Setup Control Library
- [ ] ActiveMovie control type library

OmniDriver 2.0 Type Library

Location: C:\temp\tmp4\OOI_HOME\OmniDriver32.dll

Language: Standard

► *Procedure*

To solve this problem:

1. Uncheck the reference to the obsolete OmniDriver type library.
2. Select Project | References.
3. Click the **Browse** button.
4. Navigate to OOI_HOME directory, highlight NETOmniDriver.tlb, and click **Open**.
5. Scroll down in the list of available references and check **OmniDriver .NET interface**.

6. If you are using SPAM functions, you will also need to add a reference to the new NETSpam.tlb type library following Steps 1-5.

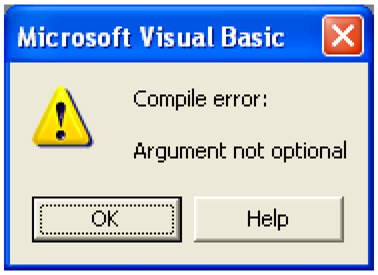
Change Dim wrapper as New OmniDriver.CCoWrapper to Dim wrapper as new CCoWrapper.

Change Dim wrapperExtensions as OmniDriver.CCoWrapperExtensions to Dim wrapperExtensions as CCoWrapperExtensions.

Make similar changes to any other OmniDriver objects which you have declared in your application.

Remove all calls to wrapper.CreateWrapper(). This method is no longer needed. However, you should continue to call the "Create" methods for SPAM objects when these "Create" methods accept one or more parameters.

You may see the following error message:



Compile error:

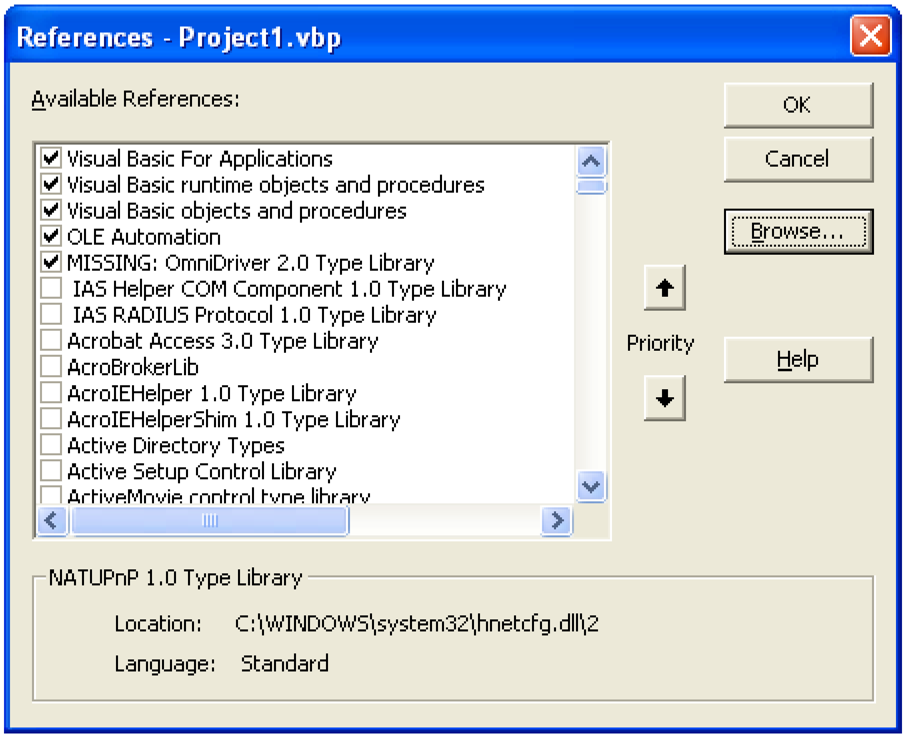
Argument not optional

To solve this error, append "_2" to the method name. Or, if the "_2" was already present, remove it.

Alternatively, you can solve this error by adding the channelIndex argument following the spectrometerIndex argument. Be sure to set the channelIndex argument to zero since this was the previous default setting that your application expects.

VB6 Applications Using the "C" Interface

If you try to make an EXE without changing the reference to the OmniDriver COM object, you will see the following screen:



Available References:

- [✓] Visual Basic For Applications
- [✓] Visual Basic runtime objects and procedures
- [✓] Visual Basic objects and procedures
- [✓] OLE Automation
- [✓] MISSING: OmniDriver 2.0 Type Library
- [ ] IAS Helper COM Component 1.0 Type Library
- [ ] IAS RADIUS Protocol 1.0 Type Library
- [ ] Acrobat Access 3.0 Type Library
- [ ] AcroBrokerLib
- [ ] AcroIEHelper 1.0 Type Library
- [ ] AcroIEHelperShim 1.0 Type Library
- [ ] Active Directory Types
- [ ] Active Setup Control Library
- [ ] ActiveMovie control type library

NATUPnP 1.0 Type Library

Location: C:\WINDOWS\system32\hnetcfg.dll\2

Language: Standard

► *Procedure*

To solve this problem:

1. Uncheck MISSING: OmniDriver 2.0 Type Library and click OK.
2. Select Project | References.
3. Click the **Browse** button.
4. Navigate to OOI_HOME directory, highlight NETOmniDriver.tlb, and click OK
5. If you are using SPAM functions, you will also need to add a reference to the NETSpam.tlb type library.

Change Dim wrapper as New OmniDriver.CCoWrapper to Dim wrapper as new CCoWrapper.

Change Dim wrapperExtensions as OmniDriver.CCoWrapperExtensions to Dim wrapperExtensions as CCoWrapperExtensions.

Make similar changes to any other OmniDriver objects which you have declared in your application.

Remove all calls to wrapper.CreateWrapper(). This method is no longer needed.
However, you should continue to call the “Create” methods for SPAM objects when these “Create” methods accept one or more parameters.

Programming the ARCoptix Spectrometer

Overview

This appendix provides information for programming the ARCoptix ANIR Series of Fourier Transform Spectrometers (FTS).

Example

Before you can use your ARCoptix spectrometer, you must download and install SpectraSuiteHub.msi installer. This is the SpectraSuite Hub Controller application. Install this software on the PC to which you intend to connect your ARCoptix unit.

► *Procedure*

1. Download the installer from the Ocean Optics Software Downloads page: **http://www.oceanoptics.com/technical/softwaredownloads.asp**. The SpectraSuite Hub Controller installer is located under Operating Software.
2. Plug your ARCoptix spectrometer into your PC.
3. After installing the SpectraSuite Hub Controller, invoke it by double-clicking on the desktop icon labeled **SpectraSuiteHub Controller**.

NOTE

If you are running on Windows 7 or later, you must right-click on the desktop icon and choose **Run as administrator**.

4. Click the **Check Status** button. If it reports "The SpectraSuite Hub service is installed but it is NOT currently running", click the **Start Service** button. Then make a note of the IP address and port number. These addresses will be needed by your application when it calls the ArcoptixTransport.connectToHub() method.

A C++ sample named vs2005_cpp_cpp_ArcoptixDemo is provided in the Sample Pack to illustrate how to control the Arcoptix spectrometer from your OmniDriver-based application.

```
        int     numberOfDiscoveredSpectrometers;
        String  serialNumber;
        boolean success;

        ArcoptixTransport anir;
        anir = new ArcoptixTransport();
        success = anir.connectToHub("10.120.19.134",7654); // replace the IP
address
        if (success == false)
        {
            System.out.println("ERROR: unable to connect to the ARCOhub");
            return;
        }
        System.out.println("We successfully connected to the ARCOhub");

        anir.nop();

        numberOfDiscoveredSpectrometers =
anir.getNumberOfDiscoveredSpectrometers();
        if (numberOfDiscoveredSpectrometers == 0)
        {
            System.out.println("No spectrometers were discovered, exiting.");
            anir.disconnect();
        }
        System.out.println("number of discovered spectrometers = " +
numberOfDiscoveredSpectrometers);

        serialNumber = anir.getSerialNumber(0);  // arbitrarily claim the
first spectrometer in the list
        success = anir.claimSpectrometer(serialNumber);
        if (success == false)
        {
            System.out.println("claimSpectrometer() failed");
            anir.disconnect();
            return;
        }
        System.out.println("claimSpectrometer() was successful");

        anir.setGain(0); // 0=low, 1=medium, 2=high, 3=extreme
        System.out.println("gain initially set to " + anir.getGain());
        anir.setGain(3); // 0=low, 1=medium, 2=high, 3=extreme
        System.out.println("gain now set to " + anir.getGain());

        double[] spectrumPixelArray;
        spectrumPixelArray = anir.getSpectrum();
        if (spectrumPixelArray == null)
        {
            System.out.println("ERROR: returned spectrum array is null");
        } else {
            System.out.println("number of pixels = " +
spectrumPixelArray.length);
            System.out.println("saturation = " + anir.getSaturationRatio());
            for (int index=0; index<10; ++index)
            {
                System.out.println("pixel[" + index + "] = " +
spectrumPixelArray[index]);
            }
        }

        System.out.println("Hit <cr> to exit");
        try {
            System.in.read();
        } catch (IOException ex) {
            // ignored
        }
        System.out.println("Now exiting");

    anir.disconnect();
```

Appendix D

Controlling Memory Size

Overview

This appendix provides information for controlling the memory size of Java and your application.

Description

By default, the Java JVM will use up to 80mb of RAM/memory. If your application requires more RAM, you can increase (or decrease) this setting by creating a file named **jvm_option.memorysize** in the OOI_HOME directory. The contents of this file should be a text string like the following: **-Xmx250m** The “250m” instructs the JVM to use up to 250 megabytes of RAM. You can set this number to higher or lower values, depending on the needs of your application and the amount of available RAM. Be careful when you do this, as an invalid option string will prevent your application from running.

Appendix E

Improving OmniDriver Performance on Windows

Overview

This appendix contains suggestions for improving the performance of OmniDriver on Windows systems.

Suggestions

Windows is not a real-time operating system and cannot guarantee a level of responsiveness. But there are a few things you can do to improve the performance of your application.

- You can increase the priority of the thread in which getSpectrum() is called. However, this only affects the priority of that thread WITHIN your application, and not relative to OTHER applications running on Windows. Often the problem is that some *other* application (or Windows service) is performing activity which interferes with the speed of your application.

  Try setting the priority of your OmniDriver application to "RealTime". To do this:

  5. Type control+alt+delete to bring up the Windows Task Manager.
  6. Select the **Processes** tab.
  7. Right-click on your OmniDriver application and choose **Set Priority | RealTime**.

- Determine what applications and services are running on your PC and shut down all unnecessary applications. If you have a backup utility such as Carbonite, you should pause it. Check your anti-virus application to see if it is configured in some way that might result in bursts of disk I/O.
- If bursts of disk I/O are interfering with your application, there is a good chance this disk I/O is due to "page faults". Page faults occur when Windows does not have enough RAM/memory to run all of the applications that are currently active. So Windows “borrows” some disk space to "simulate" additional RAM. If this causes your OmniDriver application a problem, then the solution is to either shut down as many applications as possible, or to install additional RAM.
- Try running your application on another PC that has nothing else installed.

Appendix F

Using the Timeout Feature

Overview

The Timeout feature allows you to set a timeout period on calls to getSpectrum(). This is especially useful when using external trigger modes. Formerly, when a spectrometer was set to an external trigger mode, it was impossible to escape from that mode until an actual trigger signal occurred. Using the new Timeout feature, you can regain control of the spectrometer even if a trigger signal never occurs. You set the duration of the timeout period. After getSpectrum() has waited for the full timeout period, it will return to the caller even if no trigger has occurred.

Installation

Follow the normal procedure for installing OmniDriver on your system. Then perform the steps in the following section **only if you are developing on a 32-bit Windows system**.

IMPORTANT: Converting to the WinUSB Driver (32-bit systems only)

After installing OmiDriver, some additional installation steps are necessary if you are using any of the 32-bit Windows platforms (XP, Vista, or Windows 7).

These steps are necessary because the new timeout feature is only supported by the WinUSB driver, not by the ezusb.sys driver. By default, 32-bit Windows platforms use the older ezusb.sys driver.

► ***Procedure***

1. In C:\Program Files\Ocean Optics\OmniDriverSPAM\OOI_HOME,
    a. Delete NatUSB.dll
    b. Copy or rename "NatUSB_32_winusb.dll" to "NatUSB.dll"

2. If you are using SpectraSuite on the same 32-bit PC, you must manually replace SpectraSuite's NatUSB.dll file with the NatUSB_32_winusb.dll mentioned in Step 1. Otherwise, SpectraSuite will no longer recognize any of your spectrometers. This step is only necessary on 32-bit Windows systems. To do this:
    a. cd C:\Program Files\Ocean Optics\OmniDriverSPAM\OOI_HOME

b. Copy NatUSB_32_winusb.dll to
C:\Program Files\Ocean Optics\SpectraSuite\spectrasuite\lib, renaming it to NatUSB.dll
c. Switch your spectrometer over to the WinUSB driver (follow the detailed steps below).

Manually Switching Your Spectrometer Over to the WinUSB Driver

You must perform the following procedure if you are developing on **32-bit** Windows platforms (XP, Vista, and Windows 7) because the new timeout feature is only supported by the WinUSB driver, not by the ezusb.sys driver. By default, 32-bit Windows platforms use the older ezusb.sys driver, which does not support the new timeout feature.

Notes

This solution only applies to Windows (32-bit and 64-bit architectures), not Linux or MacOSX.

This solution only works with the WinUSB driver, not with the ezusb driver. **Once you have installed WinUSB, your Windows system cannot revert back to ezusb!** If you are on a 32-bit system, and you are using SpectraSuite as well as OmniDriver, you will need to manually copy OmniDriver's NatUSB.dll into the SpectraSuite install directory. Otherwise, SpectraSuite will no longer be able to detect any spectrometers. The installation instructions explain how to perform this step.

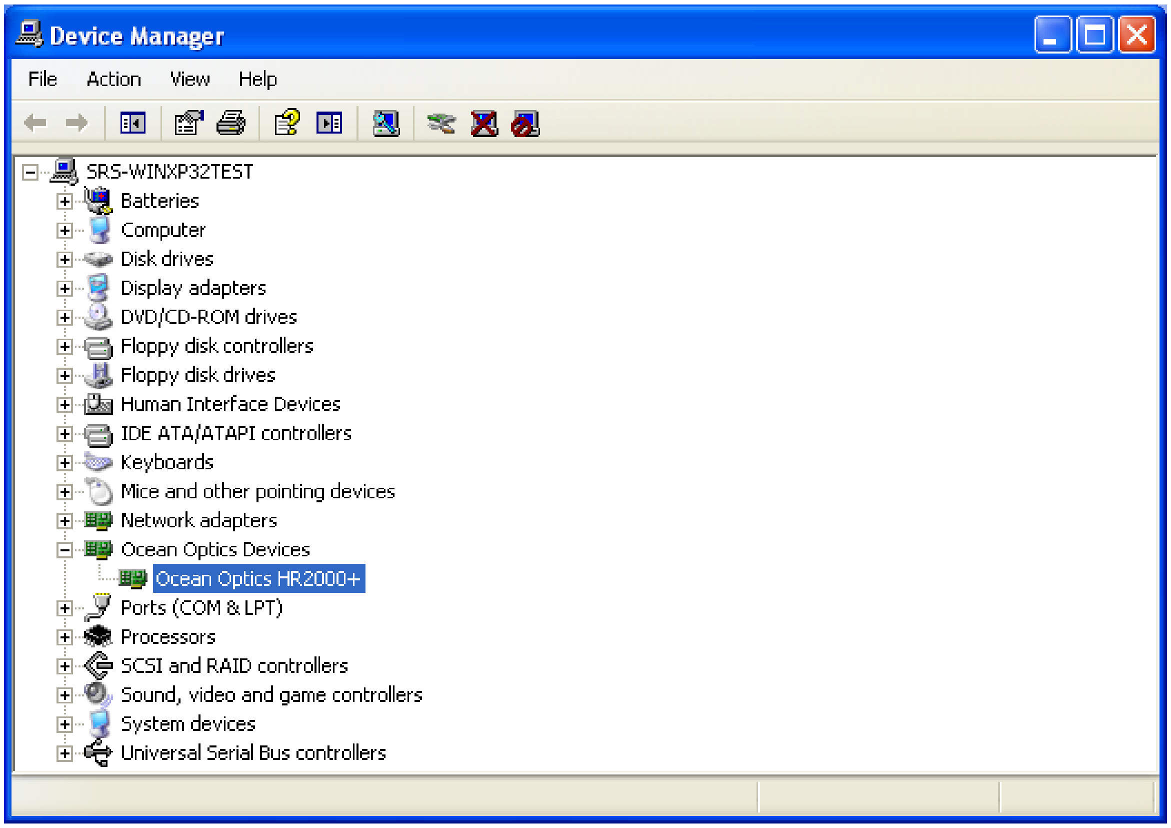
This solution only works for USB devices. It does not apply to network-attached spectrometers such as the Jaz.

The USBProgrammer utility does not work with the WinUSB driver. You cannot use this utility on systems using WinUSB. USBProgrammer only works on systems using the older ezusb driver.

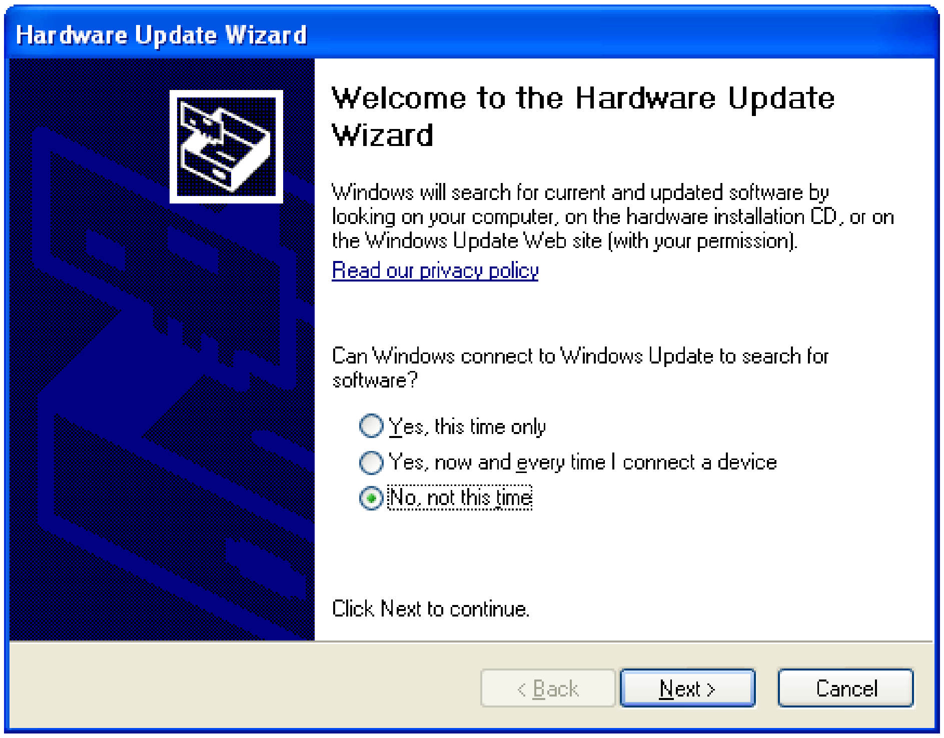
The steps shown below are for Windows XP 32-bit. The exact procedure will be slightly different for Windows Vista 32-bit and Windows 7 32-bit systems.

► *Procedure*

1. Open the Device Manager window by selecting **Control Panel | System | Hardware tab | Device Manager button**.
2. Right-click on your spectrometer and choose **Update Driver...**. The Hardware Update Wizard appears.



3. On the first screen of the Hardware Update Wizard, select **No, not this time** and click **Next**.


Hardware Update Wizard

Welcome to the Hardware Update Wizard

Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).

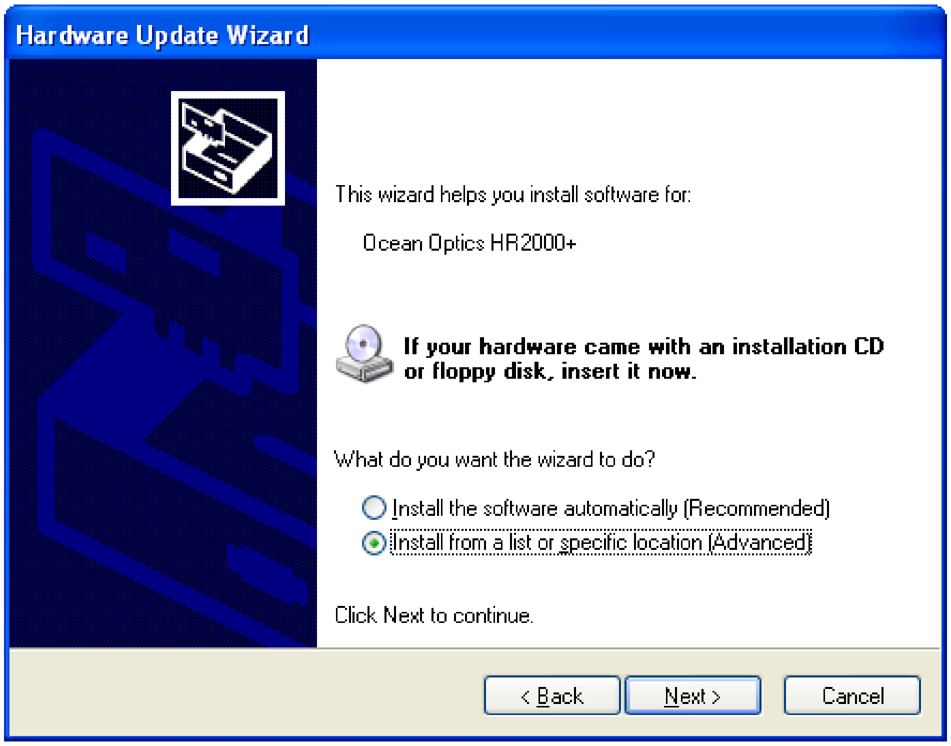
Read our privacy policy

Can Windows connect to Windows Update to search for software?

- Yes, this time only
- Yes, now and every time I connect a device
- [✓ No, not this time]

Click Next to continue.

4. On the next hardware Update Wizard screen, choose **Install from a list of specific location (Advanced)** and click **Next**.


Hardware Update Wizard

This wizard helps you install software for:

Ocean Optics HR2000+

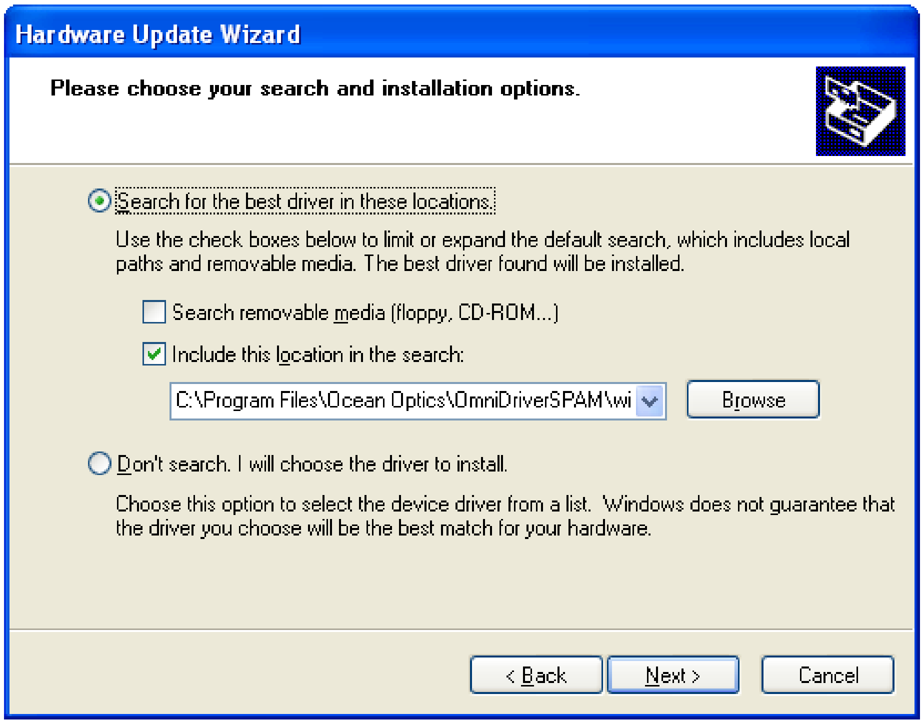
If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

○ Install the software automatically (Recommended)
[✓ Install from a list or specific location (Advanced)]

Click Next to continue.

5. Choose **Include this location in the search** and browse to C:\Program Files\Ocean Optics\OmniDriverSPAM\winusb_driver. Then click **Next**.


Hardware Update Wizard

Please choose your search and installation options.

[✓ Search for the best driver in these locations.]

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

☐ Search removable media (floppy, CD-ROM...)
☑ Include this location in the search:

C:\Program Files\Ocean Optics\OmniDriverSPAM\wi

○ Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

The software installation process begins.

Programming the Timeout Feature

To set the timeout, call wrapper.setTimeout(int timeoutMilliseconds).

To determine if a timeout has occurred, you must call wrapper.isTimeout(int spectrometerIndex, int channelIndex) after the wrapper.getSpectrum() method returns control to your application. If a timeout has occurred, the returned spectrum will be set to all zeroes, and isTimeout() will return TRUE.

Caution

If your timeout setting is shorter than the integration time, your call to getSpectrum() may time-out.

Example of Typical Usage of the Timeout Feature

When calling getSpectrum(), if a timeout occurs before your external trigger event occurs, you can simply call getSpectrum() again and continue to wait for the trigger event. Typical logic is shown below:

```
wrapper.setIntegrationTime()
wrapper.setExternalTriggerMode() // choose some external hardware
trigger mode
wrapper.setTimeout()
while (keepWaitingForTrigger() == true)  // you implement the
keepWaitingForTrigger() method
{
      wrapper.getSpectrum()
      if (wrapper.isTimeout() == true)
        continue; // timeout occurred, go to top of loop and try
      again
      // Don't forget to check for a valid spectrum result !!
      If  (wrapper.getWrapperExtensions().isSpectrumValid() ==
      false)
      {
        // Your error handler
        …
      }
      // Process valid spectrum
      …
      }
```

Notes

The timeout setting applies to all channels of the spectrometer. It is not possible to set different timeout values for each channel.

If your application uses multi-threading, you must be careful to restrict all references to a given spectrometer to a single thread. You cannot acquire spectra from the same spectrometer from multiple threads. This restriction has always been the case for OmniDriver. This means that all channels of multi-channel spectrometers must be referenced from a *single* thread. See Chapter 8: *Using OmniDriver with Multithreaded Applications* for more details.

When switching your spectrometer trigger mode from Normal mode to one of the external trigger modes, you may need to call getSpectrum() once to "flush" out the spectrum acquisition that occurred immediately prior to switching modes. This can happen because in Normal mode, the spectrometer is continuously acquiring spectra, so when you switch modes, the spectrometer will probably be in the middle of acquiring a spectrum. This spectrum will be returned by the next call you make to getSpectrum(), but this spectrum was not acquired as a result of an actual external trigger signal.

Timeout Feature Troubleshooting Notes

Problem	Possible Cause(s)	Suggested Solution(s)
Your application cannot detect any spectrometers.	A failure to copy NatUSB_32_winusb.dll to NatUSB.dll in the OOI_HOME directory.	See *IMPORTANT: Converting to the WinUSB Driver (32-bit systems only)*.
UnsatisfiedLinkError … NatUSBSetTimeout()	If using a Windows 32-bit system, you need to manually switch your computer to the newer WinUSB driver.	See *Manually Switching Your Spectrometer Over to the WinUSB Driver*.

Appendix G

Customized Branding of OmniDriver

Overview

OmniDriver versions 2.20 and later now allow you to override the company name and product name displayed during the installation process. Additionally, the folder where OmniDriver is installed will reflect the company and product name you specify.

The following is an example of the command line syntax to supply your own company and product name (all on one line):

OmniDriver-2.20-win64-redistributable-installer.exe --company_name “ACME Company” --product_name “RoadRunner Exterminator Kit”

Pay special attention – there are TWO dashes preceeding each of the new command-line parameters.

Caution

The company name and product name you specify may NOT contain embedded spaces.

It is not necessary for you to supply the new command line parameters if you want to leave company and product name set to their default values of “Ocean Optics” and “OmniDriver” respectively. If you wish, you may specify only one or the other of these two parameters.

Notes

All OmniDriver and SPAM installers have been digitally signed. Consequently, the end user will be presented with a pop-up security window indicating the installer has been signed by Ocean Optics.

Only the redistributable installers have been modified to allow you to override company and product name. The development installers do not offer this flexibility.

DLL files belonging to OmniDriver will still have names such as “OmniDriver64.dll” or “NETOmniDriver.dll.”

Index