Application Note

OSA20x Visual C++ Programming

This is a brief overview of how to get started making a custom program to communicate with an OSA20x series Optical Spectrum Analyzer. The example program is for reference only and the user is encouraged to extend or modify the program to fit his or her specific needs.

Author: Sean Tanner
Date: 16-06-2015

Part 1. Preface

This application note was written for the OSA201 Optical Spectrum Analyzer using the firmware and software versions detailed below. Functionality and procedures may vary when using other controllers or firmware/software versions.

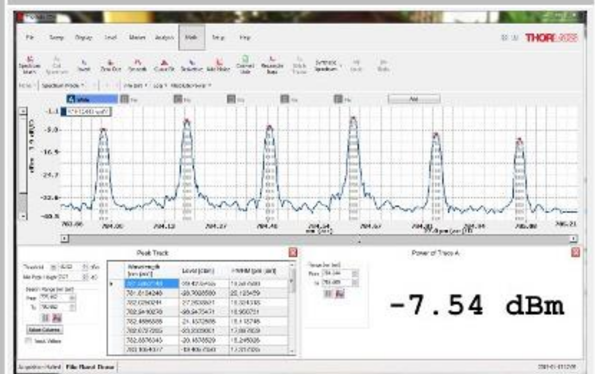
- OSA Software: Version 2.55
- OSA LabVIEW Drivers: 2.50.1249.3032
- Visual Studio: Version 11.0.61030.00 Update 4
- Microsoft .NET Framework: Version 4.5.50938

Part 2. Step by Step Instructions

1. Download and install the software for the Optical Spectrum Analyzer and Compact CCD Spectrometers located on the Software tab here: http://www.thorlabs.com/software_pages/viewsoftwarepage.cfm?code=OSA

Software | Software Updates | Libraries | Communications Protocol | Programming Reference | Archive

Software	
Description	This is a full installer for our GUI control software for our OSA and CCD spectrometers. It includes a "virtual device" mode ideal for evaluating the software. Customers who already have version 2.0 or later may download a version 2.55 updater in the *Software Updates* tab above.
Version	2.55
Filesize	419 MB
Download	Download
Change Log	Change Log
Additional	Please note: The 'Minimum System Requirements' listed below are sufficient for operating the software with a virtual device for evaluation purposes. The 'Recommended System Requirements' are strongly suggested for actual measurements.
System Requirements	Minimum • Windows Vista, 7, or 8 (32 or 64 Bit) • USB 2.0 Port • Monitor Resolution: 800 x 600 • Intel Pentium 4 or AMD 64 3000+ • 2.0 GB RAM Recommended • Windows 7 or 8 (64 Bit) • Intel Core i5 or AMD Athlon II • 6.0 GB RAM
Additional Software	.NET framework 4.0 or higher and Java Runtime 1.6 or higher are both required.

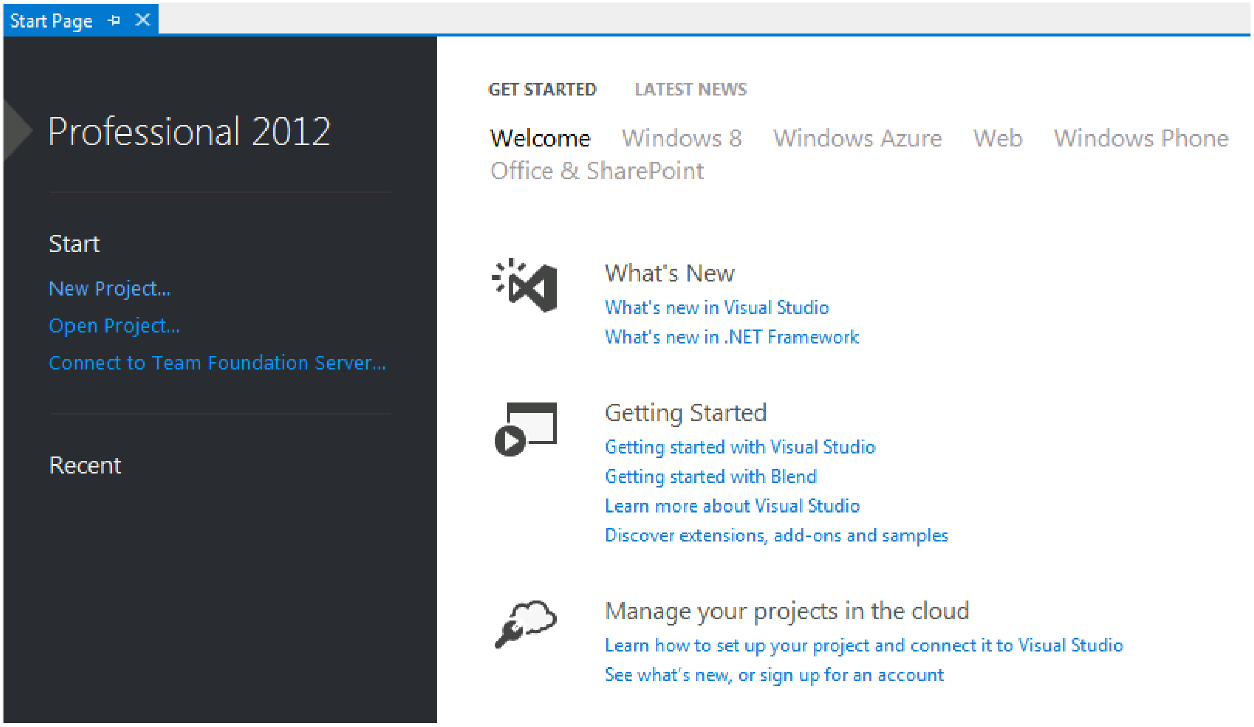


2. Connect the OSA20x to the computer via USB. Wait for Windows to install the device drivers. When the OSA20x is ready it will show up in the device manager as a Thorlabs USB Fourier Transform Spectrometer.

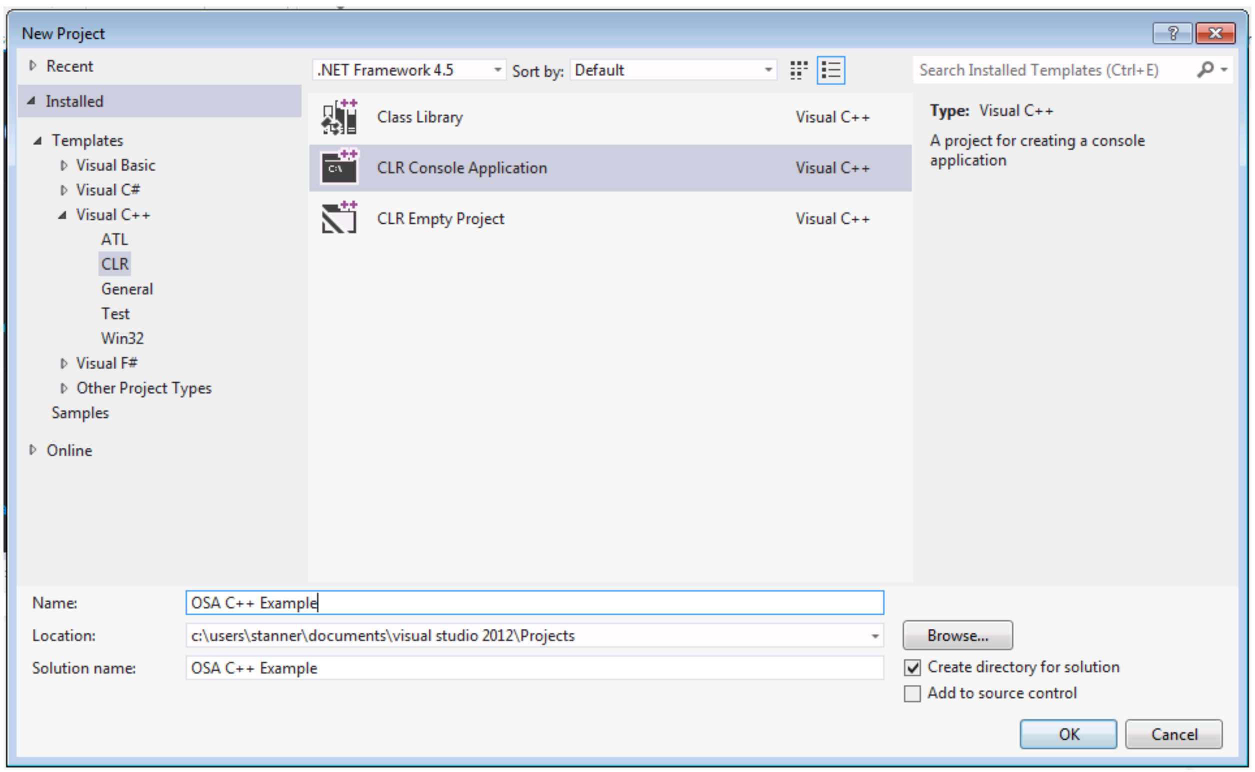
Thorlabs Devices
  Thorlabs USB Fourier Transform Spectrometer

3. Run the OSA software to become familiar with the operation of you Optical Spectrum Analyzer and to verify that it is working with the computer properly. When you are done, make sure to close the OSA software.

4. Open Microsoft Visual Studio and start a New Project.



5. Start a Visual C++ Console Application and give it an appropriate name. We used "OSA C++ Example" for this application note.



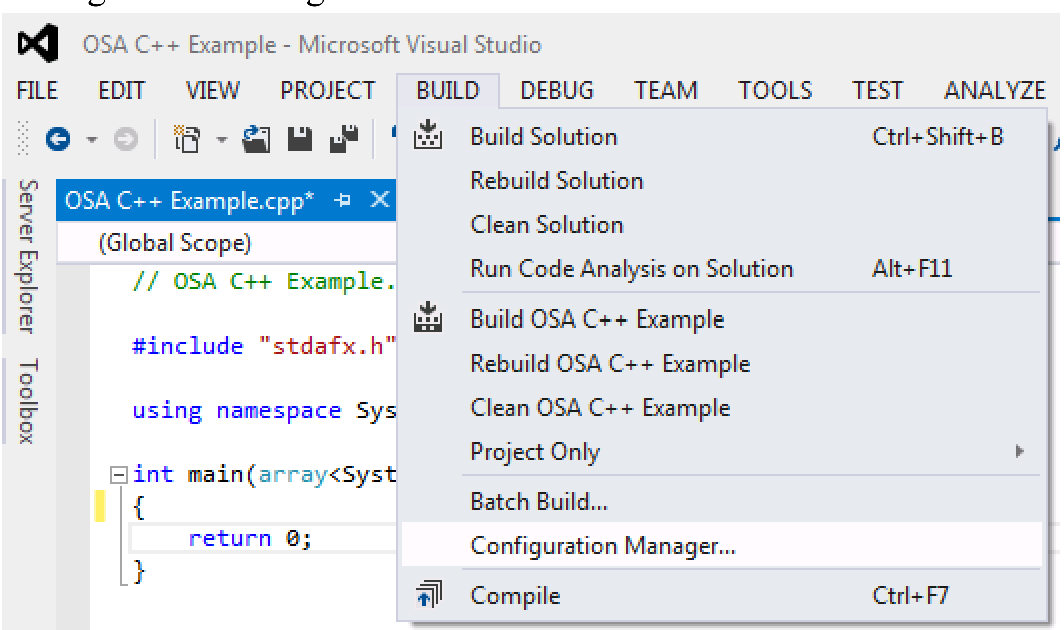
6. Your empty project will look similar to the empty project below. The only default namespace (using statement) we will use in the sample is “System”.

```
// OSA C++ Example.cpp : main project file.

#include "stdafx.h"

using namespace System;

int main(array<System::String ^> ^args)
{
    Console::WriteLine(L"Hello World");
    return 0;
}
```

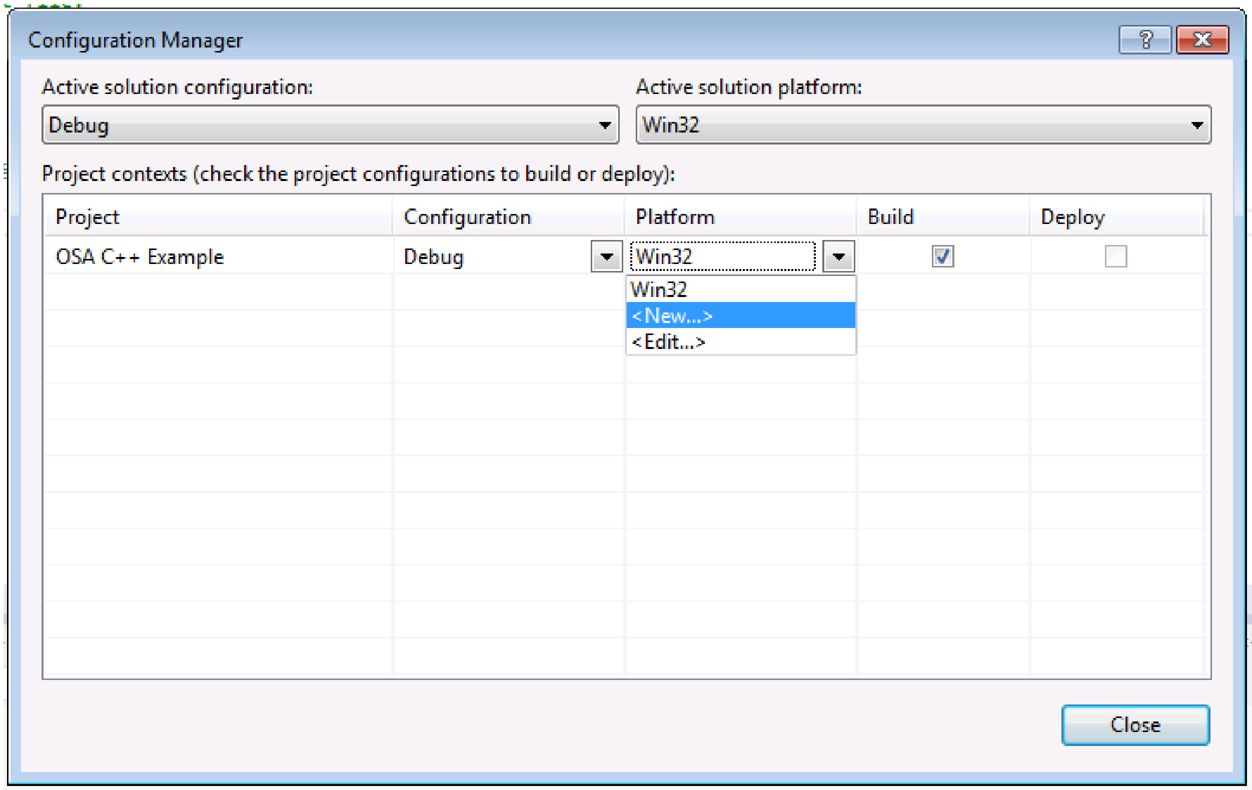
7. Remove the line that prints Hello World.

```
int main(array<System::String ^> ^args)
{
    return 0;
}
```

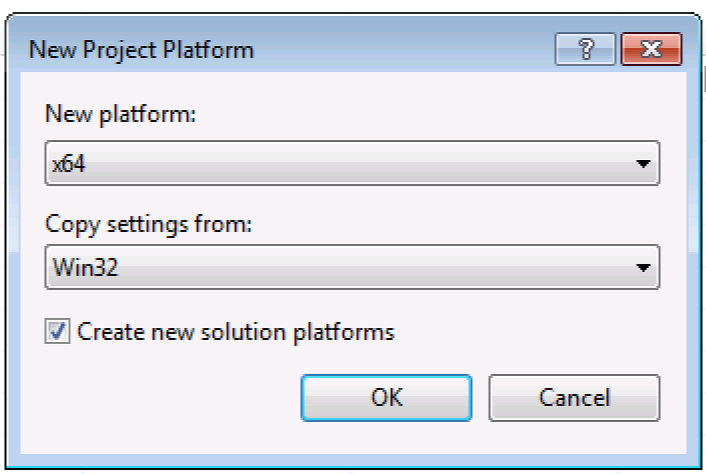
8. (Steps 8-11 are for 64-bit Windows only) From the Build menu select Configuration Manager…



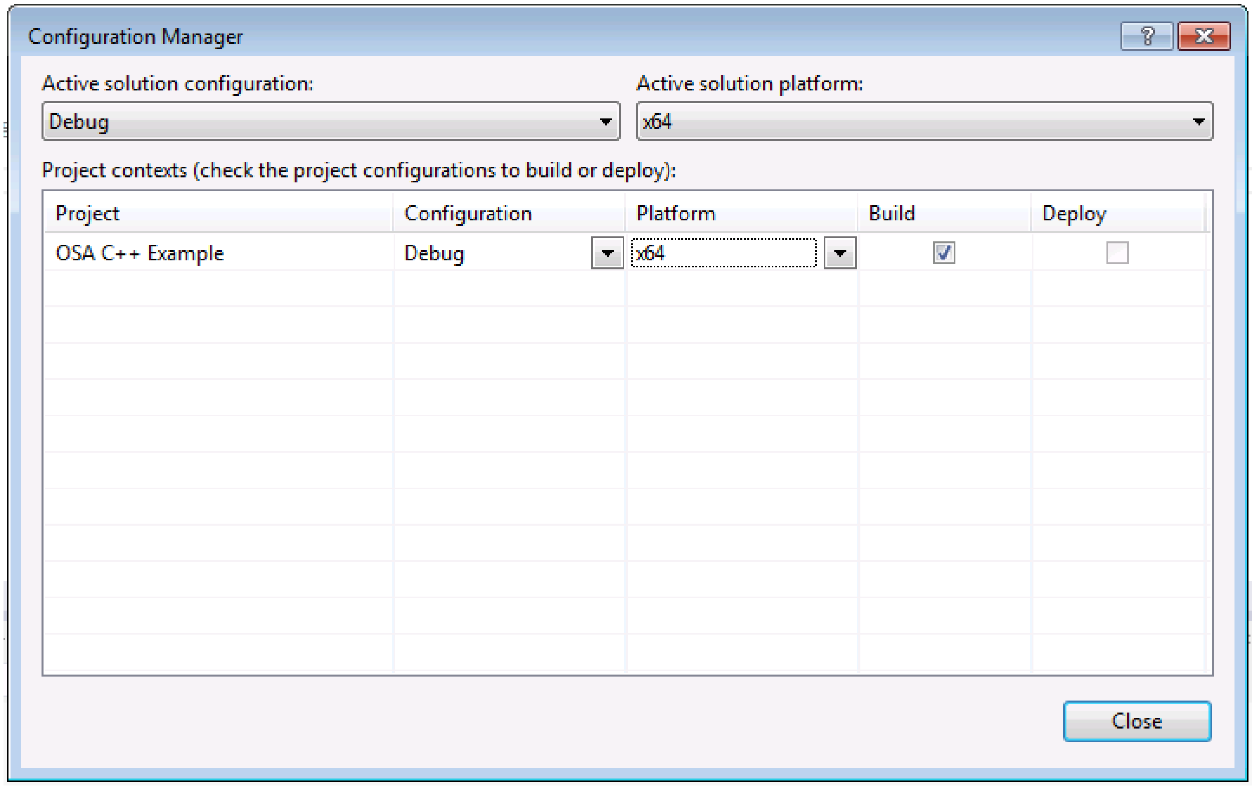
9. Open the Platform dropdown menu and click on <New…>.



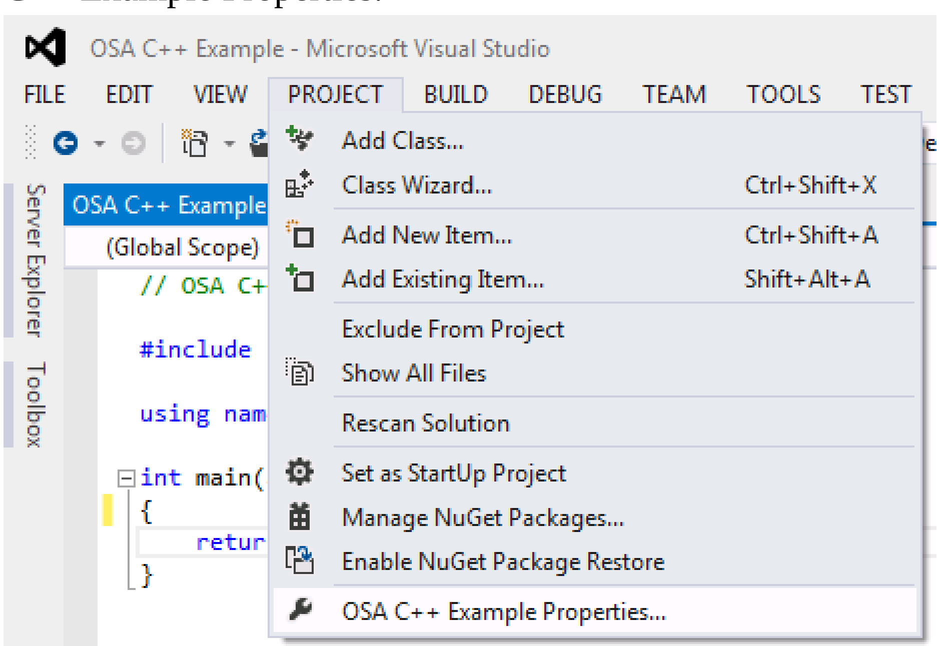
10. In the New Project Platform make sure the New platform is set to x64, Copy settings from is set to Win32 and Create new solution platforms is checked. Press OK.



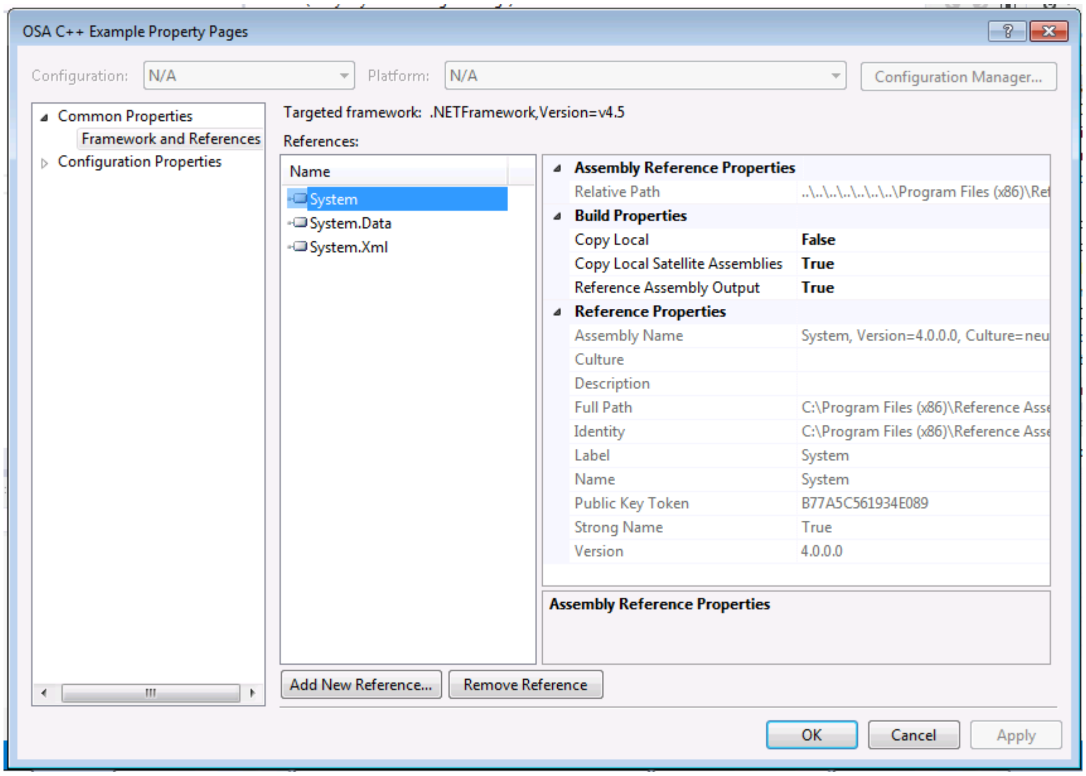
11.Close the Configuration Manager window.



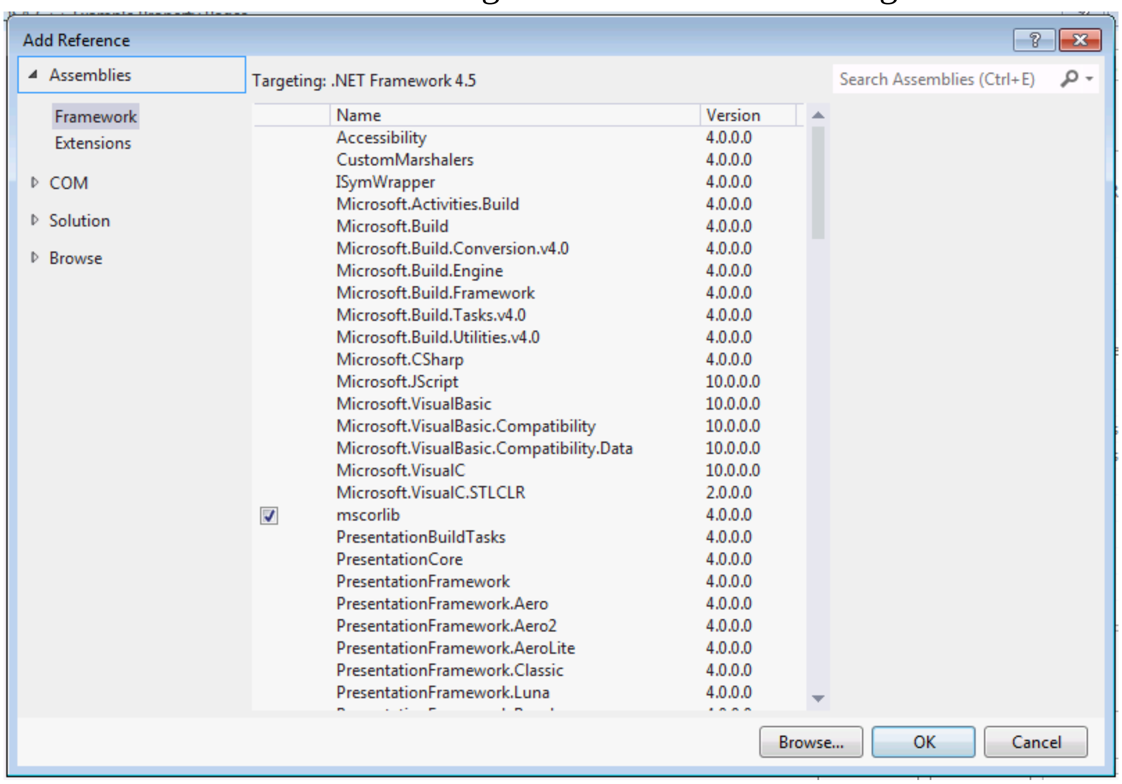
12.From the Project menu select your project name Properties, in this case OSA C++ Example Properties.



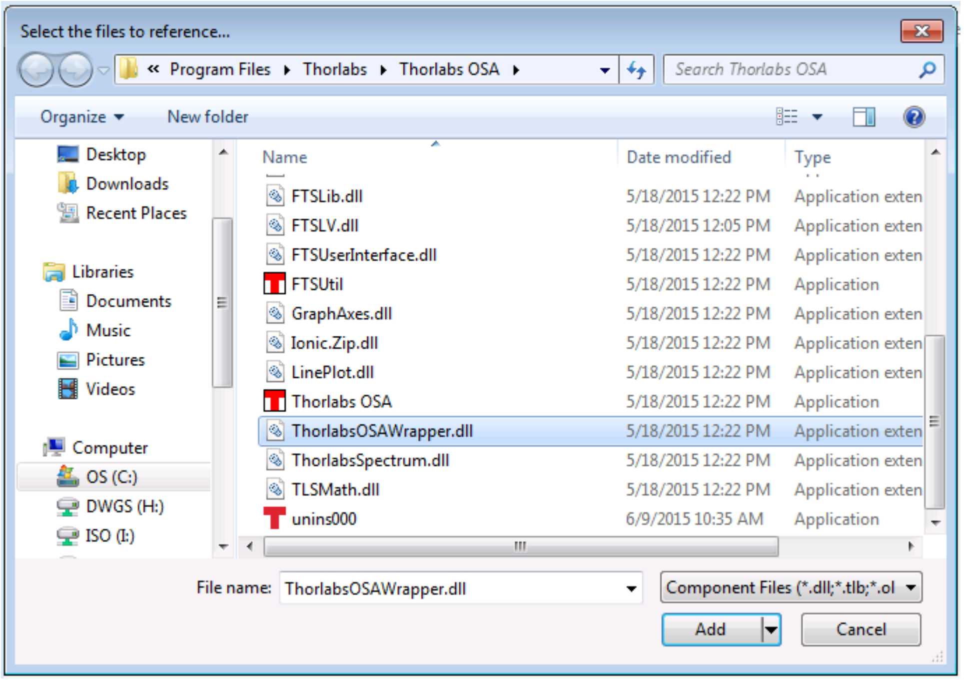
13. Select Add New Reference.



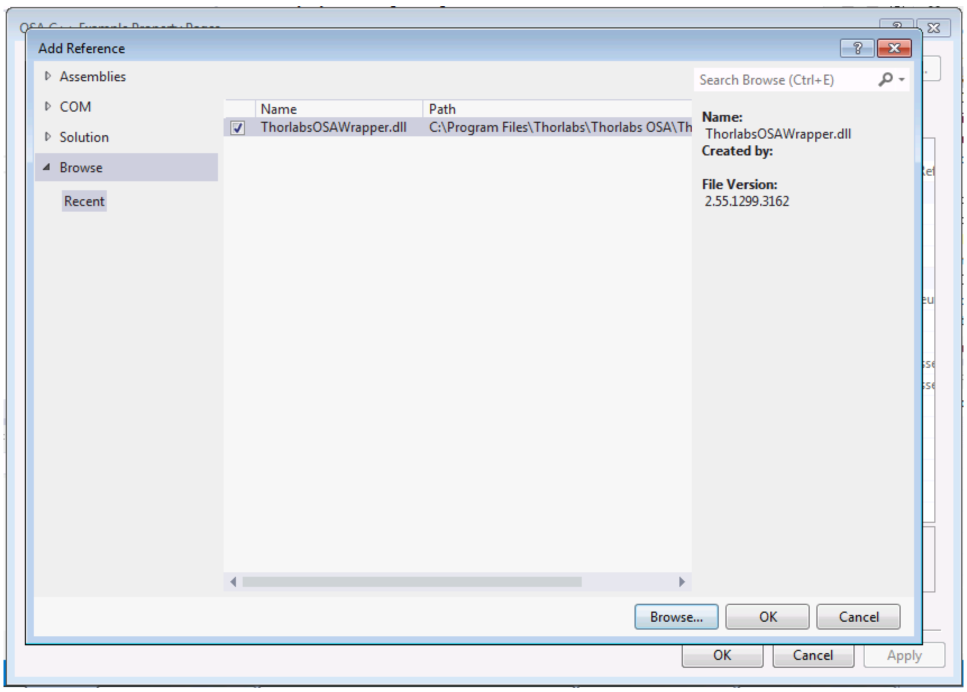
14. Select Browse from the lower right of the Reference Manager window.



15. Browse to the C:\Program Files\Thorlabs\Thorlabs OSA folder and select ThorlabsOSAWrapper.dll. Once the file is selected press Add.



16. ThorlabsOSAWrapper.dll will show up in the Reference Manager window. Make sure it is checked an press OK (for both windows).



17. Add a using statement for the ThorlabsOSAWrapper namespace.

```
// OSA C++ Example.cpp : main project file.

#include "stdafx.h"

using namespace System;
using namespace ThorlabsOSAWrapper;

int main(array<System::String ^> ^args)
{
    return 0;
}
```

18. Make the osa_OnSingle Acquisition method. This method will run when a [OnSingleAcquisition]() event is trigger by the instument. The method should be located directly after using statement for the ThorlabsOSAWrapper namespace.

```
using namespace System;
using namespace ThorlabsOSAWrapper;

static void osa_OnSingleAcquisition(System::Object^ sender, ContinuousProcessingCallbackEventArgs^ e)
{
}

int main(array<System::String ^> ^args)
```

19. Check if the data that is ready from the instrument is a spectrum. For a single acquisition the OnSingleAcquisition event will be triggered twice. Once when the interferogram is ready and again when the spectrum is ready. The next steps 20-25 will be written inside of the if statement's brackets.

```
static void osa_OnSingleAcquisition(System::Object^ sender, ContinuousProcessingCallbackEventArgs^ e)
{
    //Check if the data is a Spectrum
    if (e->UpdateFlag.Equals(AcquisitionUpdateFlag::Spectrum))
    {
    }
}
```

20. Cast the sender object (the object that triggered the event) as a [LibDeviceInterface]().

```
static void osa_OnSingleAcquisition(System::Object^ sender, ContinuousProcessingCallbackEventArgs^ e)
{
    //Check if the data is a Spectrum
    if (e->UpdateFlag.Equals(AcquisitionUpdateFlag::Spectrum))
    {
        //Cast the object that triggered the event as a LibDeviceInterface
        LibDeviceInterface^ osa = (LibDeviceInterface^)sender;
```

21. Create a new SpectrumStruct object. The constructor takes a single parameter which is the length of the spectrum. Use the GetLastSpectrumLength method to determine the length.

```
//Create a new SpectrumStruct object
//The constructor takes a single parameter which is the lenght of the spectrum
//Use the GetLastSpectrumLength() method to determine the length
SpectrumStruct^ spectrum = gcnew SpectrumStruct(osa->GetLastSpectrumLength());
```

22. Get the spectrum from the instrument. The parameter is a SpectrumStruct object in which the data is stored.

```
//Get the spectrum from the instrument
//The parameter is a SpectrumStruct object in which the data is stored
osa->GetLastSpectrum(spectrum);
```

23. Convert the x axis of the spectrum to nm in vacuum. This uses the ConvertSpectumToNanometerVac method which is a member of the UnitConversions class. The method takes two parameters. The first is the SpectrumStruct object which contains the spectrum data. The second is a boolean which indicates if the y axis should be normalized.

```
//Convert x axis of spectrum to nm in vacuum and normalize the y values
//The method takes two parameters. The first is the SpectrumStruct object which contains the data
//The second is a boolean which indicates if the y axis should be normalized
UnitConversions^ unitConverter = gcnew UnitConversions();
unitConverter->ConvertSpectrumToNanometerVac(spectrum, true);
```

24. If the spectrum is not saturated write the data to a file. The path variable should be changed to a suitable location on the users computer. The WriteSpectrum method is a member of the FileIOInterface class. It takes three parameters. The first is a SpectrumStruct object which contains the spectrum data. The second is a String which contains the file name and location. The third is an interger which defines what kind of file will be written (see methods section for more information). In the example the data is written as a text file. The \ character is an escape character which is used to place special characters in a string (\t is a tab character for example). To place a single \ in the string \\ is used.

```
//If spectrum is not saturated write the data to a file
if (!spectrum->IsSaturated)
{
    String^ path = "C:\\Users\\stanner\\Desktop\\Spectrum.txt";
    FileIOInterface^ fileWriter = gcnew FileIOInterface();
    fileWriter->WriteSpectrum(spectrum, path, 3);
    Console::WriteLine("Spectrum writen to " + path + ". Press any key to exit.");
}
```

25. If the spectrum is saturated start another acquisition. The gain will be automatically adjusted to prevent saturation, however it may take several iterations. This is the last line in the osa_OnSingle Acquisition method.

```
        else
        {
            Console::WriteLine("Spectrum Saturated");
            osa->AcquireSingleSpectrum();
        }
    }
}
```

26. Steps 27-35 will be placed in side of the Main method's brackets, before the return statement .

```
int main(array<System::String ^> ^args)
{
    return 0;
}
```

27. Call the static method DeviceLocator.[InitializeSpectrometers]() to find and initialize all OSA devices connected to the computer.

```
int main(array<System::String ^> ^args)
{
    //Find and initialize all OSAs connected to the system
    DeviceLocator::InitializeSpectrometers();
```

28. Create a new [LibDeviceInterface] object.

```
//Create a new LibDeviceInterface object
//The constructor argument is an index starting at zero for the first OSA found
LibDeviceInterface^ osa = gcnew LibDeviceInterface(0);
```

29. Set the device sensitivity.

```
//Set device sensitivity
//Argument is an unsigned short which indicates the sensitivity level
//0 = medium low sensitivity; 1 = low sensitivity
//2 = medium high sensitivity; 3 = high sensitivity
osa->SetSensitivityMode(1);
```

30. Set the device resolution.

```
//Set device resolution
//Argument is an unsigned short which indicates the resolution level
//0 = low resolution; 1 = high resolution
osa->SetResolutionMode(1);
```

31.Enable [AutomaticGain](#).

```
//Enable AutomaticGain
osa->AcquisitionSettings->AutomaticGain = true;
```

32.Register the OnSingleAcquisition event and add event handler method osa_OnSingleAcquisition. The OnSingleAcquisition event will trigger the osa_OnSingleAcquisition method (to be created later) when the instrument has finished an acquisition.

```
//Register OnSingleAcquisition event and add event handler method osa_OnSingleAcquisition
osa->OnSingleAcquisition += gcnew EventHandler<ContinuousProcessingCallbackEventArgs^>(osa_OnSingleAcquisition);
```

33.Start a single acquisition and print to the console that it has started.

```
//Start a single spectrum acquisition and let the user know it has started.
osa->AcquireSingleSpectrum();
Console::WriteLine("Single acquisition started. Press any key to cancel");
```

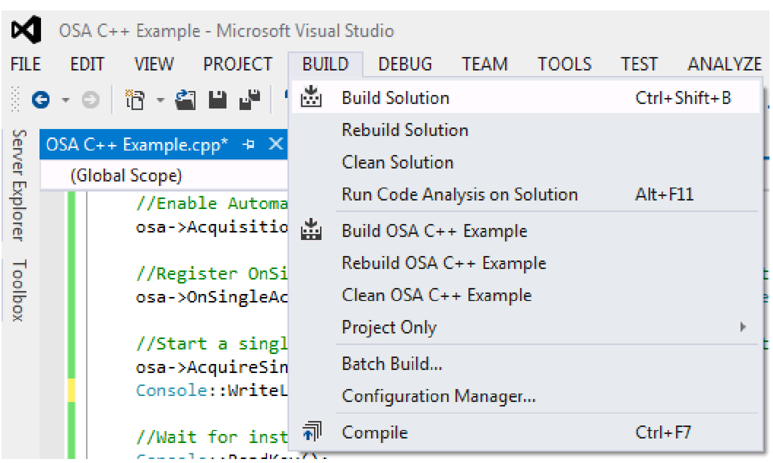
34.Wait for instrument to acquire and write spectrum.

```
//Wait for instrument to acquire and write spectrum;
Console::ReadKey();
```

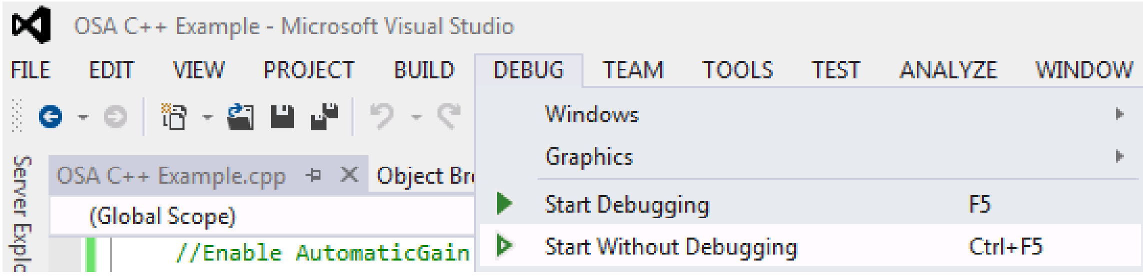
35.Close instrument. This will be the last line in the brackets of the Main method.

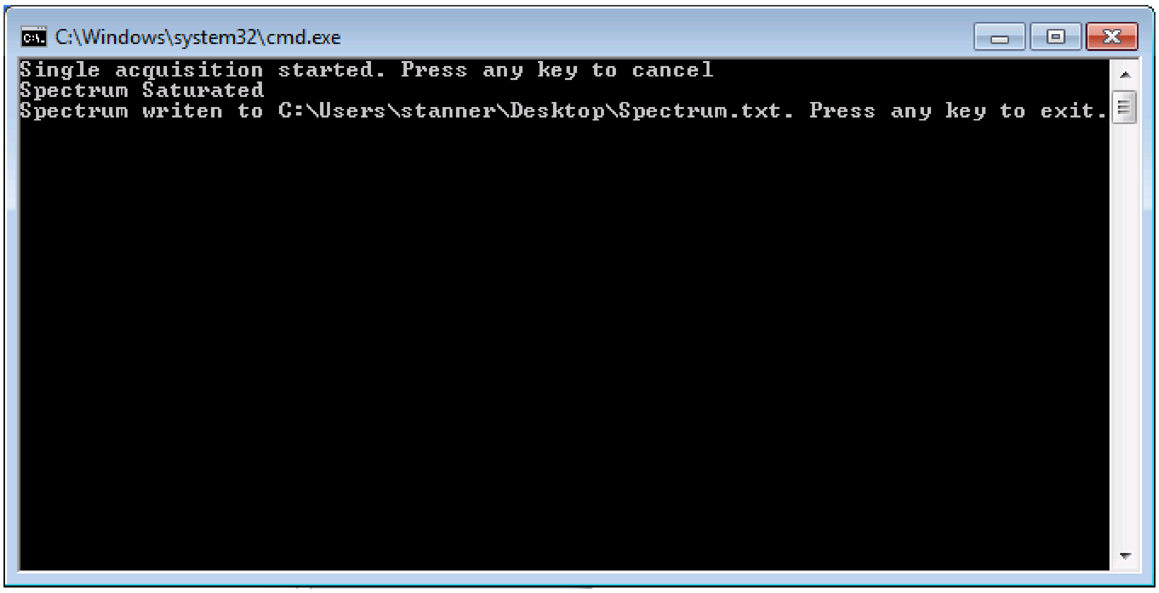
```
    //Close instrument
    osa->CloseSpectrometer();

    return 0;
}
```

36.Select Build Solution from the Build menu.



37. Select Start Without Debugging from the Debug menu to run the program.





Part 3. Methods/Properties/Events Used

public static int InitializeSpectrometers()
Member of ThorlabsOSAWrapper.DeviceLocator

Summary:
This method will scan the USB bus for all OSA units connected and initialize them.

Parameters:
None

Returns:
This function returns the number of OSA devices found as an integer.

---

public LibDeviceInterface(uint indexOfSpectrometer)
Member of ThorlabsOSAWrapper.LibDeviceInterface

Summary:
This is the constructor for the LibDeviceInterface which is used to control the OSA device.

Parameters:
indexOfSpectrometer: This parameter specifies the index of the OSA to connect to. The OSA devices found are indexed with an index starting at zero for the first OSA found.

Returns:
The function returns a new LibDeviceInterface object.

---

public virtual ThorlabsOSAWrapper.OperationResult SetSensitivityMode(ushort mode)
Member of ThorlabsOSAWrapper.LibDeviceInterface

Summary:
This function sets the OSA sensitivity.

Parameters:
Mode: An unsigned short which indicates the sensitivity level
0 = medium low sensitivity
1 = low sensitivity
2 = medium high sensitivity
3 = high sensitivity

Returns:
The function returns an OperationResult object which indicates if the sensitivity was set successfully.

---

public virtual ThorlabsOSAWrapper.OperationResult SetResolutionMode(ushort mode)
Member of ThorlabsOSAWrapper.LibDeviceInterface

Summary:
This function sets the OSA resolution.

Parameters:
Mode: An unsigned short which indicates the sensitivity level
0 = low resolution
1 = low resolution

Returns:
The function returns an OperationResult object which indicates if the resolution was set successfully.

---

bool AutomaticGain { set; get; }
Member of ThorlabsOSAWrapper.IAcquisitionSettings

Summary:
This property switches automatic gain on or off. If automatic gain is turned on the OSA will adjust the gain based on the last acquired interferogram. If the property is true automatic gain will be switched on. If the property false automatic gain will be switched off.

---

public virtual event System.EventHandler<ContinuousProcessingCallbackEventArgs> OnSingleAcquisition
Member of ThorlabsOSAWrapper.LibDeviceInterface

Summary:
This event is triggered when data is ready from a single acquisition. For a single call of the AcquireSingleSpectrum method this event will be triggered twice. Once when the interferogram is ready and once when the spectrum is ready.

The ContinuousProcessingCallbackEventArgs has three properties.
SpectrometerIndex: Index of OSA that triggered event.
StatusFlag: Information if there were any errors/warnings in the acquisition.
UpdateFlag: Information on what triggered the event (interferogram ready, spectrum ready, averaging done, acquisition canceled).

---

public virtual ThorlabsOSAWrapper.OperationResult AcquireSingleSpectrum()
Member of ThorlabsOSAWrapper.LibDeviceInterface

Summary:
This method starts the acquisition of a single spectrum. The method returns immediately. The OnSingleAcquisition event should be used to determine when the data is ready.

Parameters:
None

Returns:
The function returns an OperationResult object which indicates if the acquisition was started successfully.

---

public virtual ThorlabsOSAWrapper.OperationResult CloseSpectrometer()
Member of ThorlabsOSAWrapper.LibDeviceInterface

Summary:
This method closes the connection with the current OSA Device.

Parameters:
None

Returns:
The function returns an OperationResult object which indicates if the device was closed successfully.

---

public SpectrumStruct(uint length)
Member of ThorlabsOSAWrapper.SpectrumStruct

Summary:
This is the constructor for the SpectrumStruct which is used to store spectrum/interferogram data.

Parameters:
Length: This parameter specifies number of data points in the spectrum. The GetLastSpectrumLength method can be used to get the number of data points in the last acquired spectrum.

Returns:
The function returns a new SpectrumStruct object.

---

public virtual uint GetLastSpectrumLength()
Member of ThorlabsOSAWrapper.LibDeviceInterface

Summary:
This method gets the number of data points in the last acquired spectrum.

Parameters:
None

Returns:
The function returns the number of data points as a uint.

---

public virtual ThorlabsOSAWrapper.OperationResult GetLastSpectrum(ThorlabsOSAWrapper.SpectrumStruct spec)
Member of ThorlabsOSAWrapper.LibDeviceInterface

Summary:
This method gets the data from the last acquired spectrum.

Parameters:
Spec: A SpectrumStruct object in which the spectral data will be stored.

Returns:
The function returns an OperationResult object which indicates if the spectrum was retrieved successfully.

---

public virtual ThorlabsOSAWrapper.OperationResult ConvertSpectrumToNanometerVac(ThorlabsOSAWrapper.SpectrumStruct spectrum, bool normalize)
Member of ThorlabsOSAWrapper.UnitConversions

Summary:
This method converts the x-axis of spectrum data into nanometers in vacuum. It will also optionally normalize the y-axis.

Parameters:
Spectrum: A SpectrumStruct object in which contains the spectral data. The converted data is stored in the same object.
Normalize: A Boolean value which indicates is the y-axis should be normalized.

Returns:
The function returns an OperationResult object which indicates if the spectrum was converted successfully.

---

public bool IsSaturated { set; get; }
Member of ThorlabsOSAWrapper.SpectrumStruct

Summary:
This property indicates if the OSA detector was saturated.

---

public virtual ThorlabsOSAWrapper.OperationResult WriteSpectrum(ThorlabsOSAWrapper.SpectrumStruct spec, string fileNameAndPath, int fileFormat)
Member of ThorlabsOSAWrapper.FileIOInterface

Summary:
This method writes spectrum or interferogram data to a file.

Parameters:
Spec: A SpectrumStruct object in which contains the spectral data.
fileNameAndPath: A string which contains the full file name and path of the file that will be written. If the file does not exist it will be created. If the file does exist it will be overwritten.
fileFormat: An integer which indicates what kind of file should be created. Note only Comma Separated Values and Thorlabs OSA Spectrum Files store all header information.

0 = SPC spectrum file format
1 = Comma Separated Values
2 = Thorlabs OSA Spectrum File
3 = Raw text file, no header
4 = Matlab v5 binary data format
5 = Zipped Comma Separated Values File
6 = Zipped Raw text file, no header
7 = JCampDX ASCII data file

Returns:
The function returns an OperationResult object which indicates if the data was written successfully.

Part 4. Full Program

```
// OSA C++ Example.cpp : main project file.

#include "stdafx.h"

using namespace System;
using namespace ThorlabsOSAWrapper;

static void osa_OnSingleAcquisition(System::Object^ sender,
ContinuousProcessingCallbackEventArgs^ e)
{
	//Check if the data is a Spectrum
	if (e->UpdateFlag.Equals(AcquisitionUpdateFlag::Spectrum))
	{
		//Cast the object that triggered the event as a LibDeviceInterface
		LibDeviceInterface^ osa = (LibDeviceInterface^)sender;

		//Create a new SpectrumStruct object
		//The constructor takes a single parameter which is the lenght of the
spectrum
		//Use the GetLastSpectrumLength() method to determine the length
		SpectrumStruct^ spectrum = gcnew SpectrumStruct(osa-
>GetLastSpectrumLength());

		//Get the spectrum from the instrument
		//The parameter is a SpectrumStruct object in which the data is stored
		osa->GetLastSpectrum(spectrum);

		//Convert x axis of spectrum to nm in vacuum and normalize the y values
		//The method takes two parameters. The first is the SpectrumStruct object
which contains the data
		//The second is a boolean which indicates if the y axis should be
normalized
		UnitConversions^ unitConverter = gcnew UnitConversions();
		unitConverter->ConvertSpectrumToNanometerVac(spectrum, true);

		//If spectrum is not saturated write the data to a file
		if (!spectrum->IsSaturated)
		{
			String^ path = "C:\\Users\\stanner\\Desktop\\Spectrum.txt";
			FileIOInterface^ fileWriter = gcnew FileIOInterface();
			fileWriter->WriteSpectrum(spectrum, path, 3);
			Console::WriteLine("Spectrum writen to " + path + ". Press any key
to exit.");
		}
		//If the spectrum is saturated start another acquisition.
		//This may take several itterations.
		else
		{
			Console::WriteLine("Spectrum Saturated");
			osa->AcquireSingleSpectrum();
		}
	}
}

int main(array<System::String ^> ^args)
```

```
{
        //Find and initialize all OSAs connected to the system
    DeviceLocator::InitializeSpectrometers();

    //Create a new LibDeviceInterface object
    //The constructor argument is an index starting at zero for the first OSA found
    LibDeviceInterface^ osa = gcnew LibDeviceInterface(0);

    //Set device sensitivity
    //Argument is an unsigned short which indicates the sensitivity level
    //0 = medium low sensitivity; 1 = low sensitivity
    //2 = medium high sensitivity; 3 = high sensitivity
    osa->SetSensitivityMode(1);

    //Set device resolution
    //Argument is an unsigned short which indicates the resolution level
    //0 = low resolution; 1 = high resolution
    osa->SetResolutionMode(1);

    //Enable AutomaticGain
        osa->AcquisitionSettings->AutomaticGain = true;

    //Register OnSingleAcquisition event and add event handler method
osa_OnSingleAcquisition
    osa->OnSingleAcquisition += gcnew
EventHandler<ContinuousProcessingCallbackEventArgs^>(osa_OnSingleAcquisition);

    //Start a single spectrum acquisition and let the user know it has started.
    osa->AcquireSingleSpectrum();
    Console::WriteLine("Single acquisition started. Press any key to cancel");

    //Wait for instrument to acquire and write spectrum;
    Console::ReadKey();

    //Close instrument
    osa->CloseSpectrometer();

    return 0;
}
```

Part 5. Other Resources

The Help file for the ThorlabsOSAWrapper.dll can be found in C:\Program Files\Thorlabs\Thorlabs OSA\DotNet.

The Help file for FTSlib.lib can be found in C:\Program Files\Thorlabs\Thorlabs OSA\lib.

Commented C style header files for the library can be found in C:\Program Files\Thorlabs\Thorlabs OSA\include.

A C# example is available in the Start Menu in All Programs>Thorlabs>Thorlabs OSA>OSA>DotNet>CSharp Example.

The Object Browser in Visual Studio will list all of the classes and methods available in the ThorlabsOSAWrapper.dll. The Object Browser can be opened from the View menu in Visual Studio once a reference to ThorlabsOSAWrapper.dll has been added (steps 11-14).