LC-MISDK
Reference Manual

[Version 1.00]

• Official application names used in this manual

Abbreviated in this manual as:	Official name
Windows 10	Microsoft® Windows® 10 Pro Operating System
Windows 11	Microsoft® Windows® 11 Pro Operating System

• Trademarks

Microsoft and Windows are the trademarks of Microsoft Corporation in the United States and/or other countries. All other corporate and product names mentioned in this manual are properties of their respective owners.

• Notes on this manual

The contents of this manual are subject to change without prior notice.
Notwithstanding the preceding Konica Minolta assumes no liability for any result obtained from the use of this manual.

Contents

1. Overview 8

1.1 Foreword 8
1.2 Definitions (Terms, Abbreviations) 8
1.3 Operating Environment 8
1.4 Supported Instruments 8

2. Setup 9

2.1 Installing the NuGet Package 10
1. Installation Procedure 10
2. Uninstall 17
2.2 Basic Usage 19
1. Specify the namespace 19
2. Create the variables to which the SDK instance and the return value class will be assigned. 19
3. Start communication with the instrument 19
4. Call the APIs and check for errors using the return values 20
5. End communication with the instrument 21

3. CS/LS-150, CS/LS-160 22

3.1 Overview 22
3.2 Notice 22
3.3 Use Cases 22
1. Execute a measurement 23
2. Read the measurement data stored in the instrument 26
3. Change the backlight setting 29
4. Change the measurement time setting 31
5. Run single-point calibration 34
6. Run RGB calibration 39
3.4 Supported API List (CS-150/CS-160) 45
3.5 Supported API List (LS-150/LS-160) 48
3.6 Color Space 50

4. API Specification 51

4.1 API List 51
4.2 API Format 54

API specification format 54
Class: ReturnMessage 55
Color Space Class 56
4.3 Communication 57
Connect() 57
DisConnect() 58
GetDeviceList() 59
4.4 Measurement 60
Measure() 60
Enum: MeasStatus 61
PollingMeasurement() 61
CancelMeasurement() 62
4.5 Current Measurement Data 63
ReadLatestData() 63
ReadDisplayValue() 64
4.6 Measurement Data (Stored in Device) 65
GetNumberOfSampleData() 65
ReadSampleData() 66
DeleteSampleData() 71
4.7 Target Values 72
SetTargetCh() 72
GetTargetCh() 73
ReadTargetData() 74
WriteTargetData() 75
DeleteTargetData() 76
4.8 Measurement Time/Synchronization Settings 77
Class: MeasurementTime 77
Enum: MeasTimeMode 77
SetMeasurementTime() 78
GetMeasurementTime() 80
Class: MeasurementFrequency 81
Enum: SyncMode 81
SetSyncMode() 82
GetSyncMode() 84
4.9 Peak/Valley 85
Enum: PeakValley 85

SetPeakValley() 85
GetPeakValley() 86
4.10 LensSettings 87
Enum: CloseUpLensType 87
SetCloseUpLens() 88
GetCloseUpLens() 89
4.11 User Calibration Channel 90
SetCalibrationCh() 90
GetCalibrationCh() 91
4.12 User Calibration 92
Class: UserCalibData 92
Enum: CalibType 92
SetMatrixCalib() 93
GetCalibData() 96
DeleteCalibData() 98
4.13 CCF (Color Correction Factor) 99
Class: ColorCorrectionFactor 99
Enum: CCFMode 99
SetCCF() 100
GetCCF() 101
4.14 Power Settings 102
Enum: AutoPowerOff 102
SetAutoPowerOff() 102
GetAutoPowerOff() 103
4.15 Backlight 104
Enum: BackLightMode 104
SetBackLightOnOff() 104
GetBackLightOnOff() 105
Enum: BackLightLevel 106
SetBackLightLevel() 107
GetBackLightLevel() 108
4.16 Display Digits 109
SetColorDispDigit() 109
GetColorDispDigit() 110
4.17 Display Type (Absolute Value/Difference/Ratio) 111
Enum: DispType 111

SetDisplayType() 112
GetDisplayType() 113
4.18 Language/Date/Time 114
Enum: DisplayLanguage 114
SetDisplayLanguage() 114
GetDisplayLanguage() 115
Class: DateTime 116
SetDateTime() 116
GetDateTime() 118
Enum: DateFormat 119
SetDateFormat() 119
GetDateFormat() 120
4.19 Device Color Mode 121
Enum: ColorMode 121
SetColorMode() 122
GetColorMode() 123
Enum: ColorModeDisplay 124
SetColorModeDisplayOnOff() 124
GetColorModeDisplayOnOff() 126
4.20 Data Save Mode 127
Enum: DataSaveMode 127
SetDataSaveMode() 127
GetDataSaveMode() 128
4.21 Periodic Calibration Settings 129
Enum: PeriodicCalibNotify 129
SetPeriodicCalibNotify() 129
GetPeriodicCalibNotify() 131
4.22 Button Settings 132
Enum: ToggleStatus 132
SetToggleOnOff() 133
GetToggleOnOff() 134
Enum: TriggerStatus 135
SetTriggerOnOff() 135
GetTriggerOnOff() 137
4.23 Device/SDK Information 138
Class: DeviceInfo 138

GetDeviceInfo()....139
GetSDKVersion()....140
4.24 Measurement Unit Settings....141
Enum: LuminanceUnit....141
SetLuminanceUnit()....141
GetLuminanceUnit()....142

5. Appendix....143

5.1 Error Code List....143
5.2 Color Space Class....144
Class: MeasurementData....144
Class: XYZ....145
Class: Lvxy....146
Class: Lvudvd....148
Class: LvTcpDuv....150
Class: LvDwPe....152
Class: Lv....154
5.3 Device driver install....155
Automatic installation....155
Manual installation....155

1. Overview

1.1 Foreword

The LC-MISDK is a tool for developing PC application software for instruments that measure light source color. This manual describes how to use the LC-MISDK. This manual assumes that the application developer will be using Microsoft Visual C#, so the programming methods described here use Microsoft Visual C#.

1.2 Definitions (Terms, Abbreviations)

Terms, abbreviations	Meaning
Current measurement value	The latest measurement data stored in the instrument.
Remote mode	The mode where the instrument is communicating with a PC and the buttons are disabled.
Connection/connection status	Starting communication with the SDK, the communicating status.

1.3 Operating Environment

Supported operating systems	Windows 10 (32-bit/64-bit) Windows 11
Operating environment	.NET Framework 4.5
Development environment *	Visual Studio 2013 Visual Studio 2015
Development language	Visual C#
Supported platforms	x64 AnyCPU

* An environment connected to the Internet is required to install the necessary external packages.

1.4 Supported Instruments

CS/LS-150 Series	CS-150, CS-160 LS-150, LS-160

2. Setup

To use the LC-MISDK, you must first create a new C# application in Visual Studio and then install the LC-MISDK NuGet package.

		Description
LC-MISDK_win100_all/	nupkg/	NuGet package file
	SampleProgram/	C# sample program solution
	Manual/	Reference Manual
	driver/	USB communication driver
	license/	EULA (end-user license agreement, package license information)

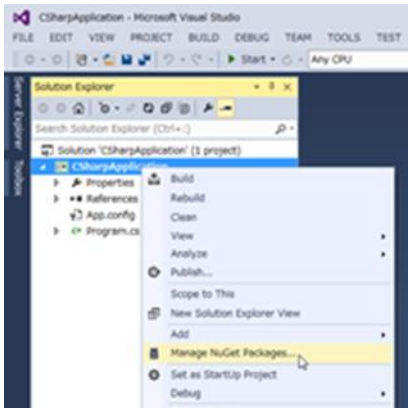
2.1 Installing the NuGet Package

Install the LC-MISDK using the NuGet tool included with Visual Studio 2013 and subsequent versions. The DLLs and configuration files required for application development are copied to the project and the references are automatically set by installing the NuGet package.

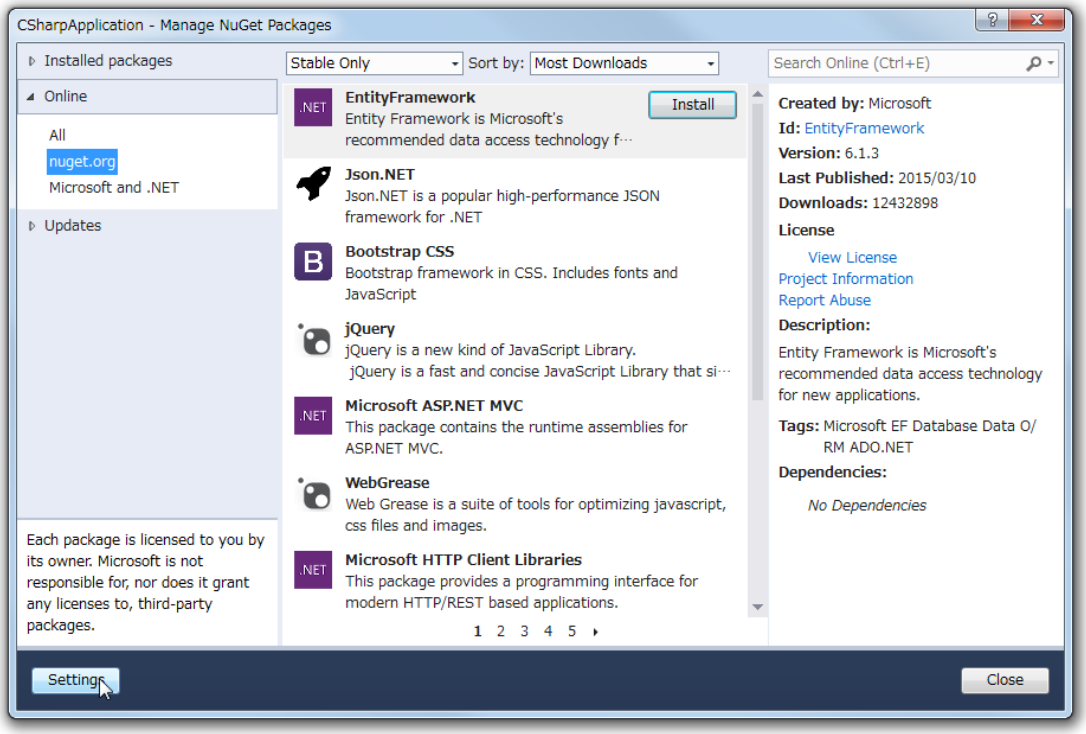
1. Installation Procedure

---In the case of Visual Studio 2013---

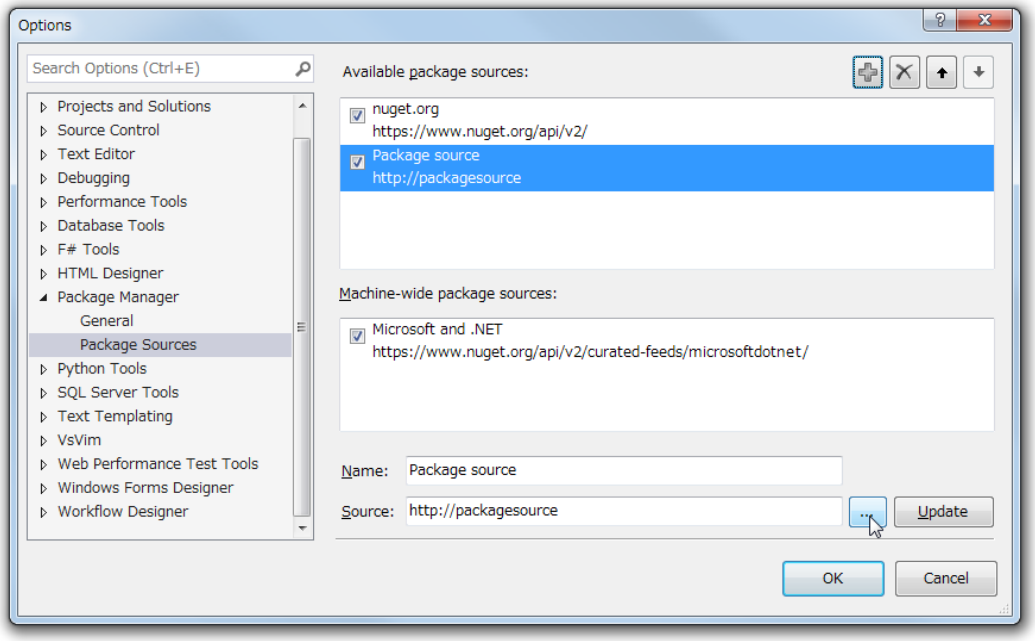
Right-click the project and select **Manage NuGet Packages** to open Manage NuGet Packages menu.



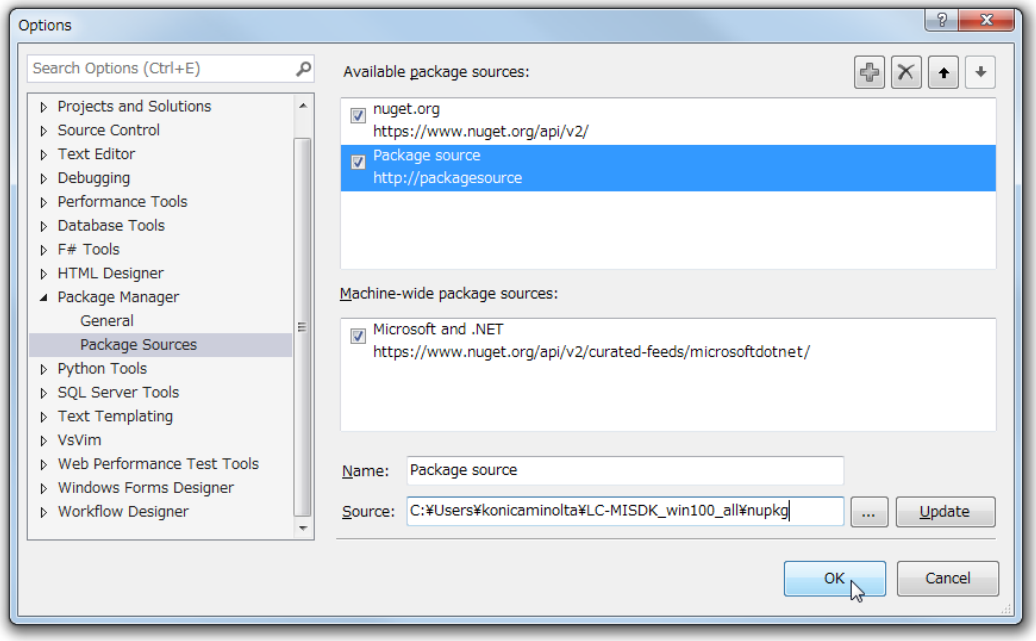
Click **Settings** button to open **Options** menu.



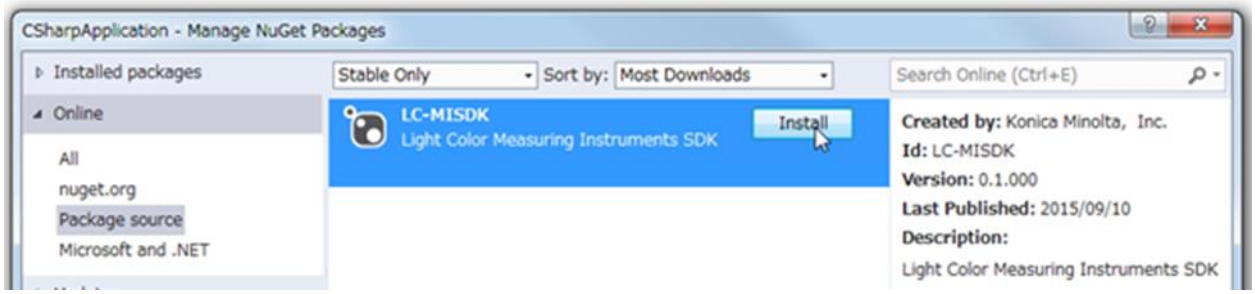
Click **Add** button at the upper right to create a new package source. Next click "..." to specify the location of the package source.



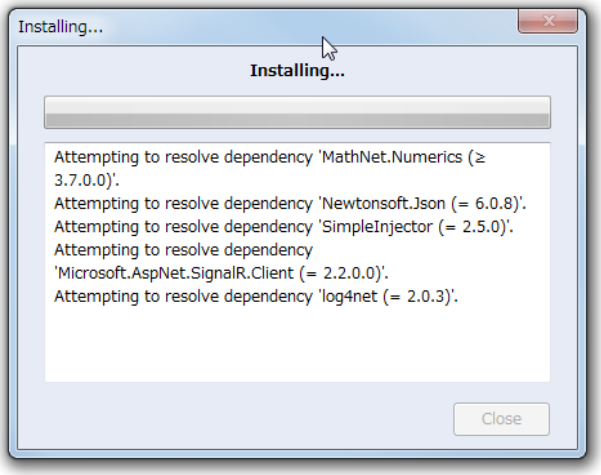
Specify the folder (LC-MISDK_win100_all/nupkg/) where CL-MISDK.nupkg is located as the location of the package source. When the folder has been specified in Source, click **OK** button.



LC-MISDK is displayed in **Manage NuGet Packages** under **Online**. Select LC-MISDK and click **Install** button.

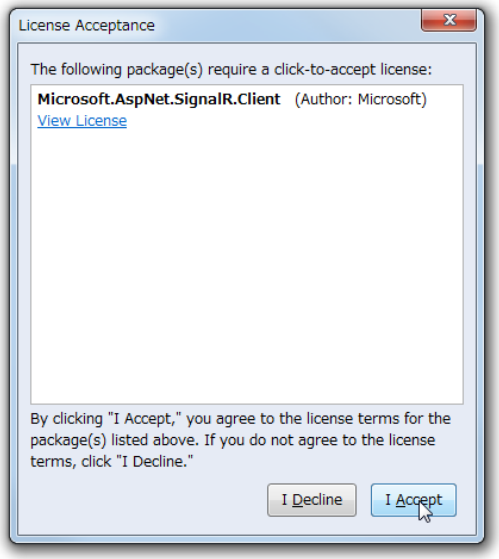


The installation will begin.

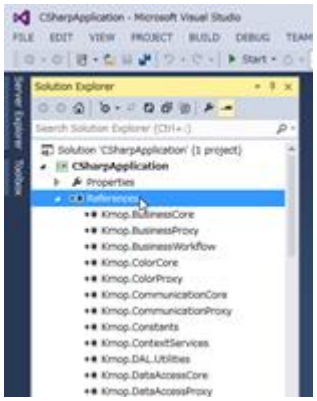


* An environment that is connected to the Internet is required because the external package will be downloaded and installed from nuget.org.

On **License Acceptance** dialog box, select "**I Accept**".

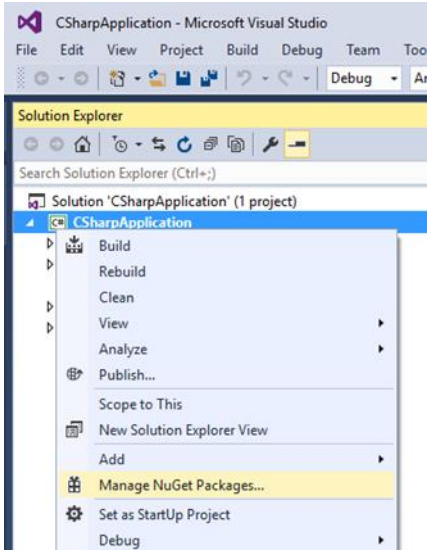


The required DLLs are referenced and the installation will be completed.

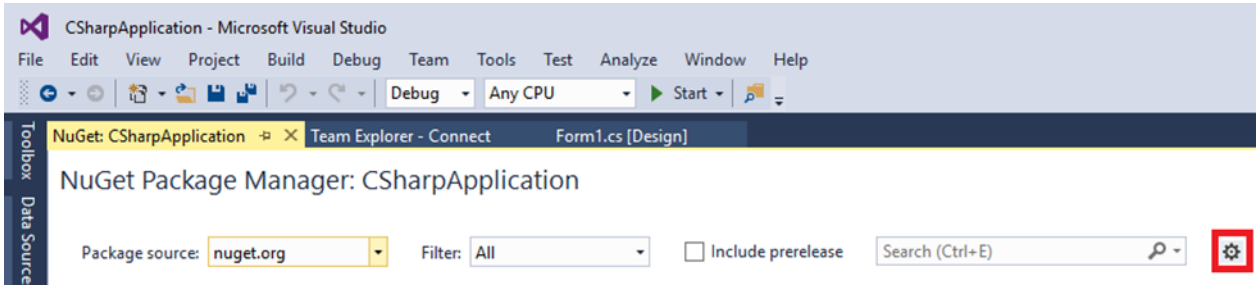


---In the case of Visual Studio 2015---

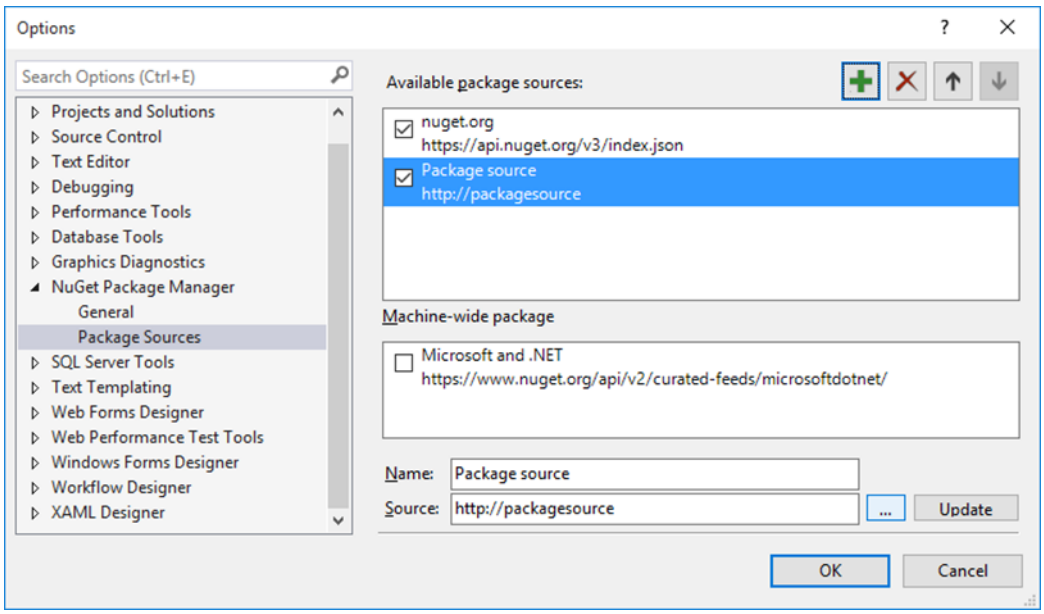
Right-click the project and select **Manage NuGet Packages** to open NuGet Package Manager.



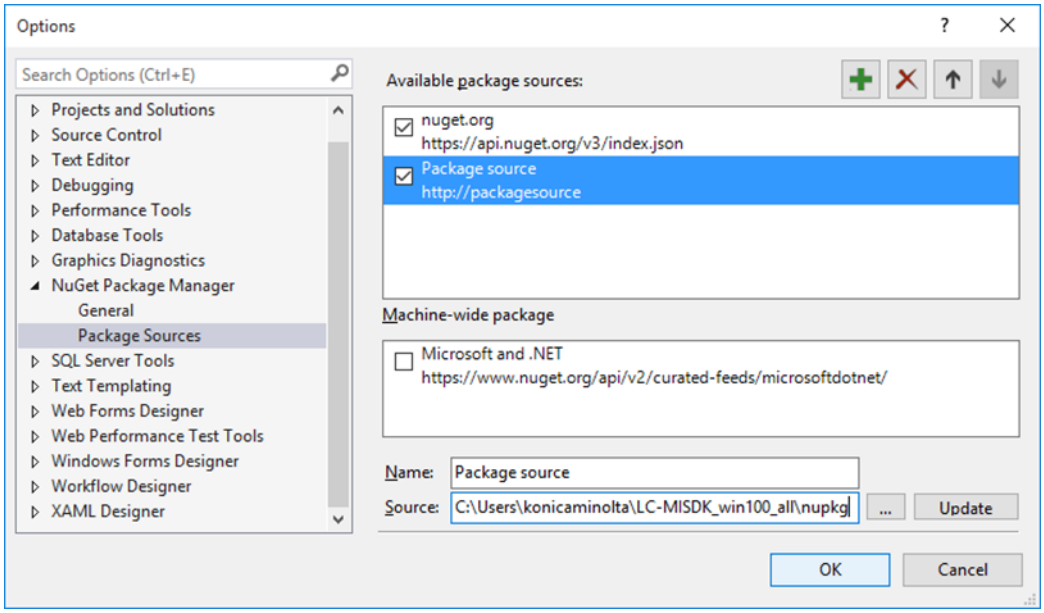
Click **Settings** icon to open **Options** menu.



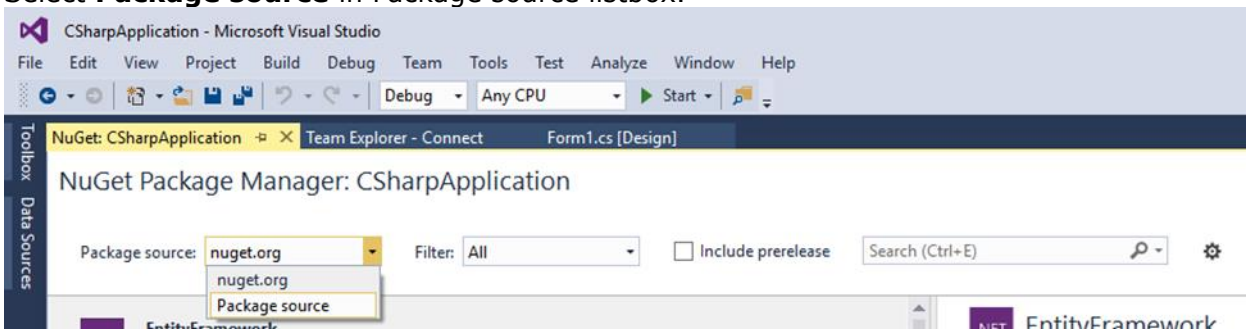
Click **Add** button at the upper right to create a new package source. Next click "..." to specify the location of the package source.



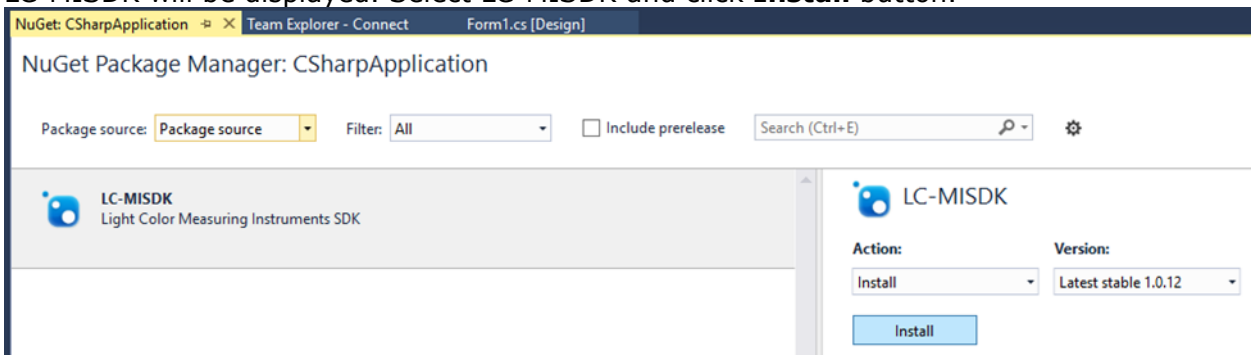
Specify the folder (LC-MISDK_win100_all/nupkg/) where LC-MISDK.nupkg is located as the location of the package source. When the folder has been specified in Source, click **OK** button.



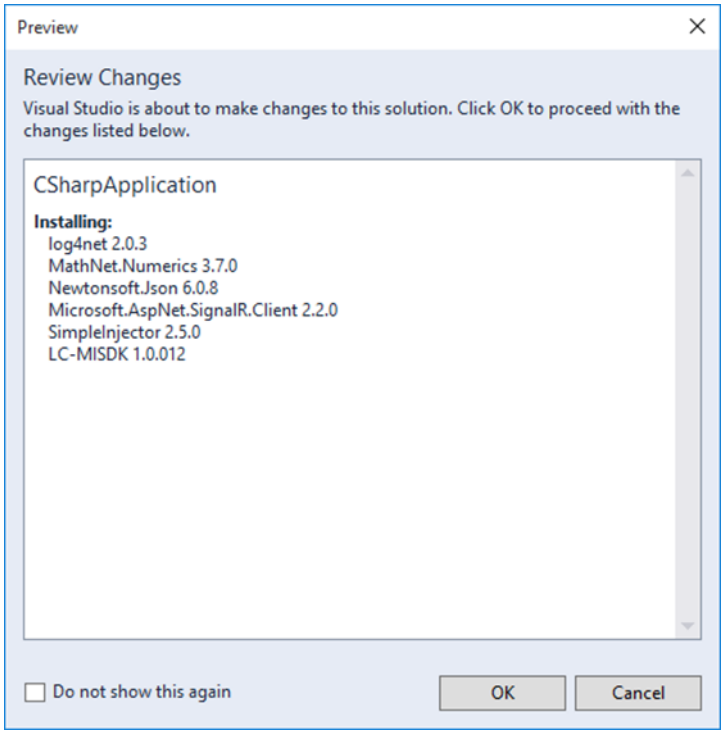
Select **Package source** in Package source listbox.



LC-MISDK will be displayed. Select LC-MISDK and click **Install** button.

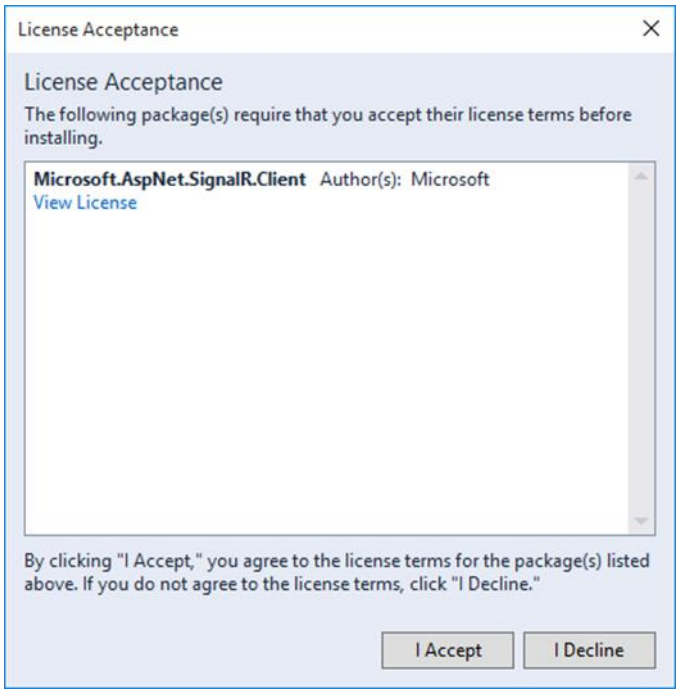


Click **OK** button on **Review Changes** dialog box. The installation will begin.

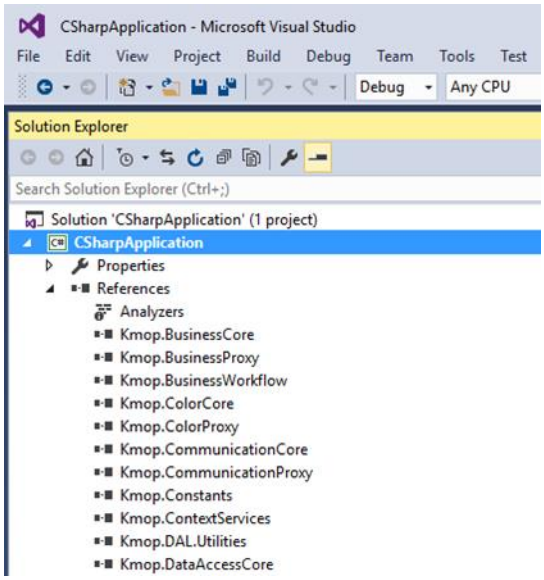


* An environment that is connected to the Internet is required because the external package will be downloaded and installed from nuget.org.

On **License Acceptance** dialog box, select "**I Accept**".



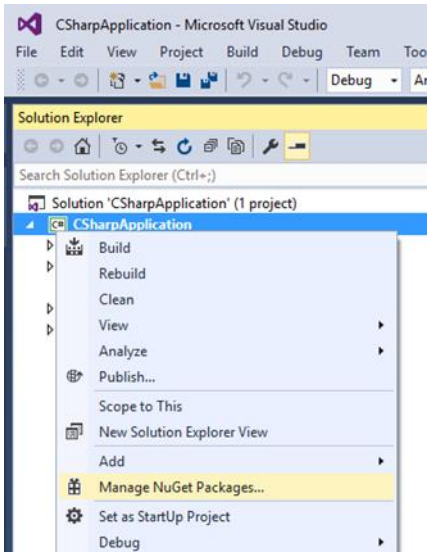
The required DLLs are referenced and the installation will be completed.



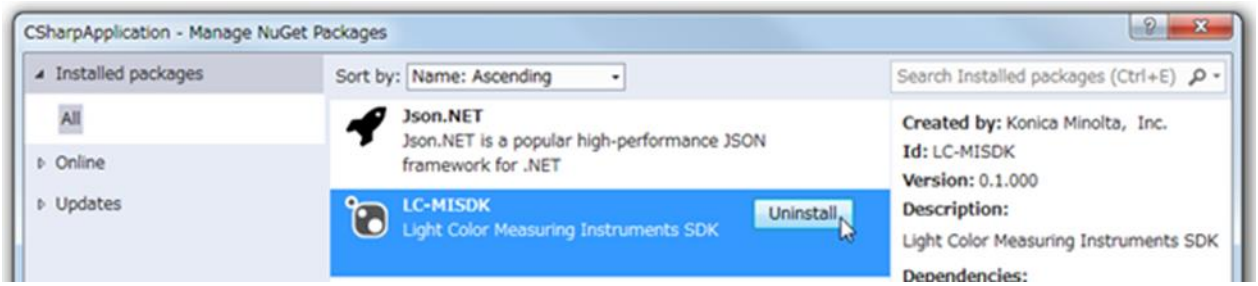
2. Uninstall

---In the case of Visual Studio 2013---

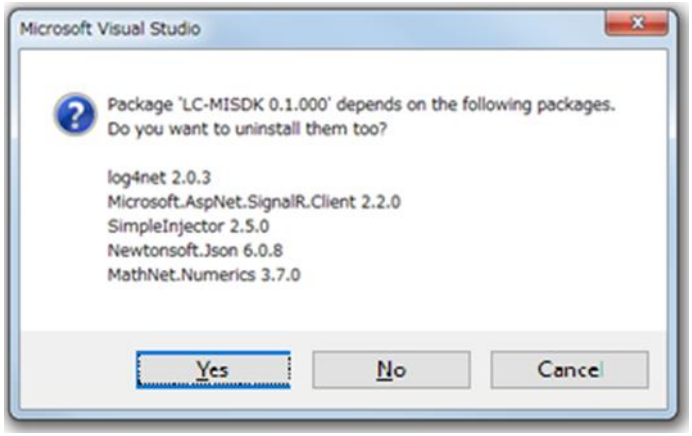
Right-click the project where the SDK has been installed and open **Manage NuGet Packages** menu.



LC-MISDK is displayed in **Manage NuGet Packages** under **Installed packages**. Select LC-MISDK and click **Uninstall** button.



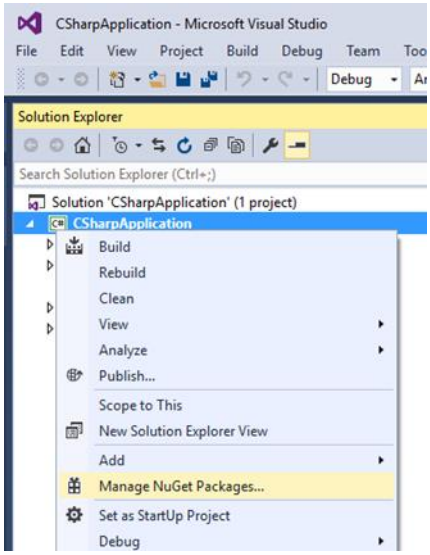
If the dialog box to confirm uninstalling the external package is displayed, click **Yes**.



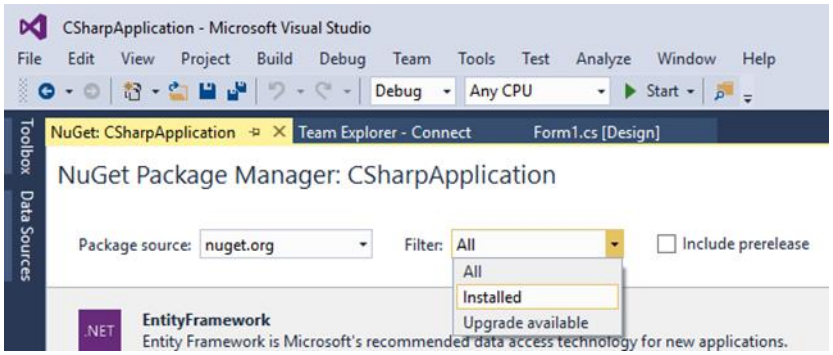
The SDK references are reset and the referenced files are deleted to finish the uninstallation.

---In the case of Visual Studio 2015---

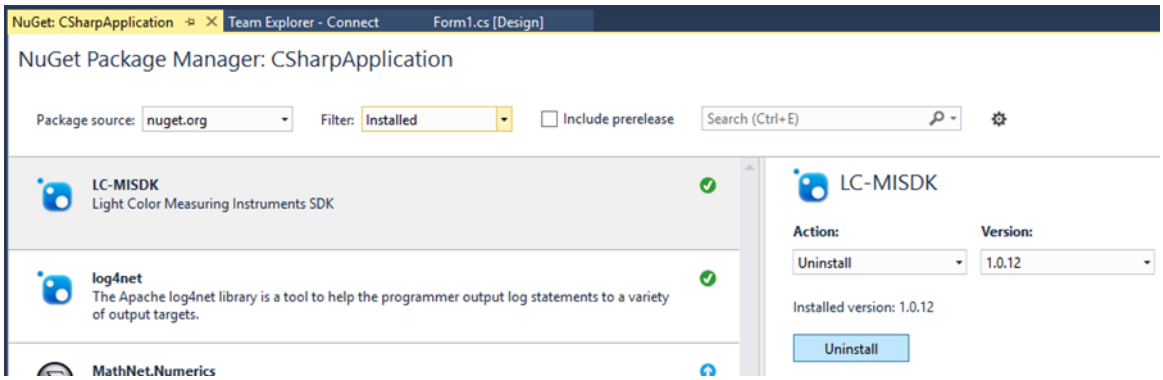
Right-click the project where the SDK has been installed and open **NuGet Package Manager**.



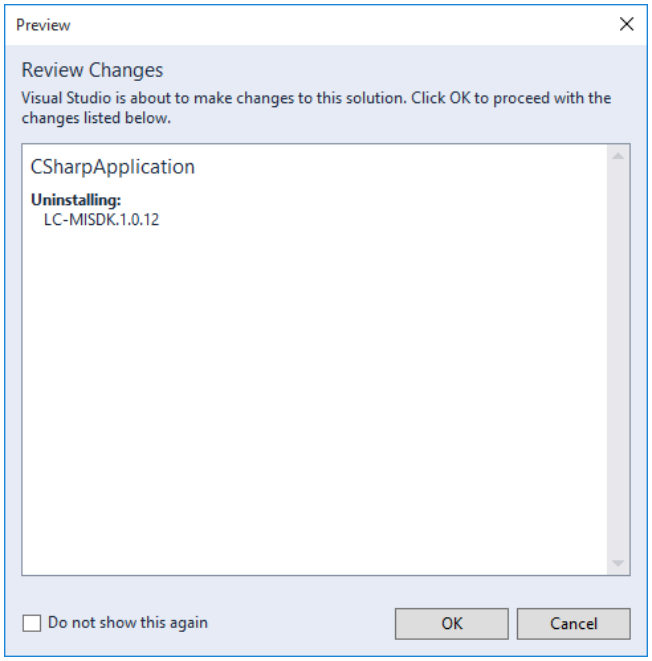
Select **Installed** in Filter listbox.



LC-MISDK will be displayed. Select LC-MISDK and click **Uninstall** button.



Click **OK** button on **Review Changes** dialog box.



The SDK references are reset and the referenced files are deleted to finish the uninstallation.

2.2 Basic Usage

To use the LC-MISDK, first create an instance of the SDK, and then call the APIs from that instance. It is not possible to launch multiple applications referring to a common LC-MISDK.dll simultaneously, since only a single instance of the SDK can be created. Before using the SDK functions, connection to the instrument must be established using the connection API. This process is described below.

1. Specify the namespace

```
using Konicaminolta;
```

2. Create the variables to which the SDK instance and the return value class will be assigned.

```
LightColorMISDK sdk = LightColorMISDK.GetInstance();
ReturnMessage ret;
```

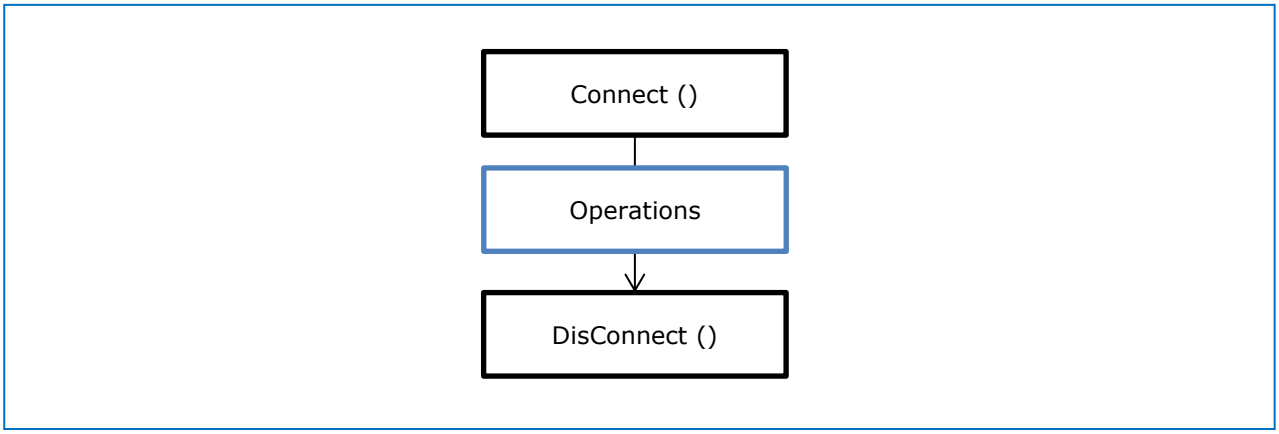
3. Start communication with the instrument

Use the created instance to call the API.

```
ret = sdk.Connect();
```

4. Call the APIs and check for errors using the return values

Use Connect() to connect to the instrument. Call the APIs and execute the SDK functions in this state. Lastly, end the connection to the instrument with DisConnect().



Error handling using API return values

```
ret = sdk.Connect();
if( ret.errorCode != ErrorDefine.KmSuccess )
{
        // error handling codes
}
```

5. End communication with the instrument

```
ret = sdk.DisConnect();
```

3. CS/LS-150, CS/LS-160

3.1 Overview

The CS-150 and CS-160 are handheld meters for measuring color and luminance. The LS-150 and LS-160 are handheld luminance meters. The LC-MISDK can configure, take measurements, and calibrate these instruments from a PC. Since the CS-150 and CS-160 can measure both color and luminance while the LS-150 and LS-160 can measure only luminance, the functions that can be used for each meter differ somewhat. (3.3, 3.4)

3.2 Notice

If any of the following actions are performed while the instrument is connected to the PC, the connection API "Connect()" may cease to function in some cases. If that occurs, please switch off the instrument power and then switch it back on, and then try to connect to the instrument again.

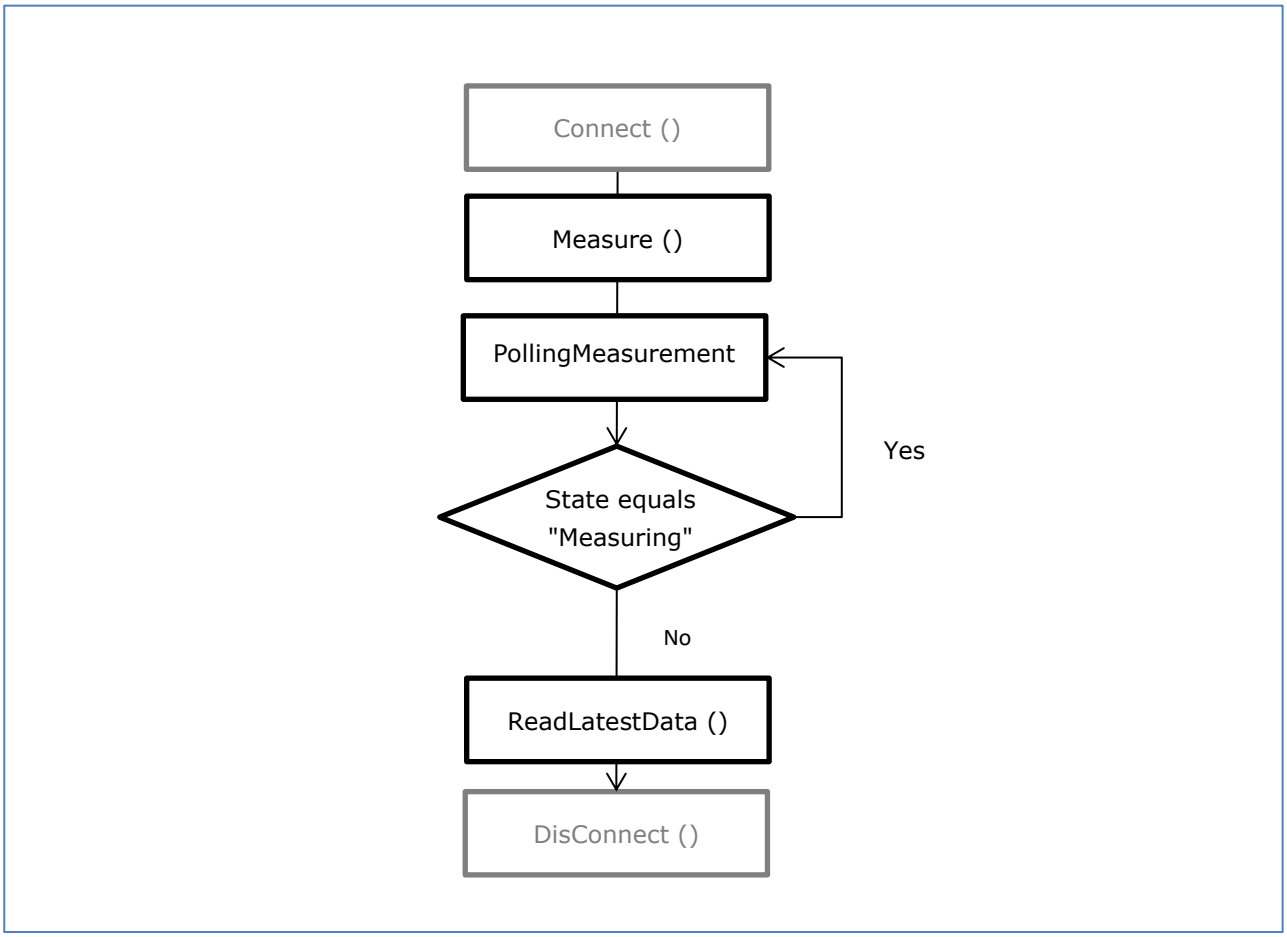
* Booting, rebooting, or hibernating Windows
* Switching off the power or unplugging the instrument without calling the disconnection API

3.3 Use Cases

Measurements	
1	Execute a measurement (Measure → Obtain the measurement value in a certain color space).
2	Read the measurement data stored in the instrument.
Configuration	
3	Change the backlight setting.
4	Change the measurement time setting.
Calibration	
5	Run single-point calibration.
6	Run RGB calibration.

1. Execute a measurement

ID	Use case name		Overview
1	Execute a measurement from the SDK and obtain the measurement value.	1	Start a measurement.
		2	Keep polling the instrument and wait for the measurement to end.
		3	Obtain the measurement value.



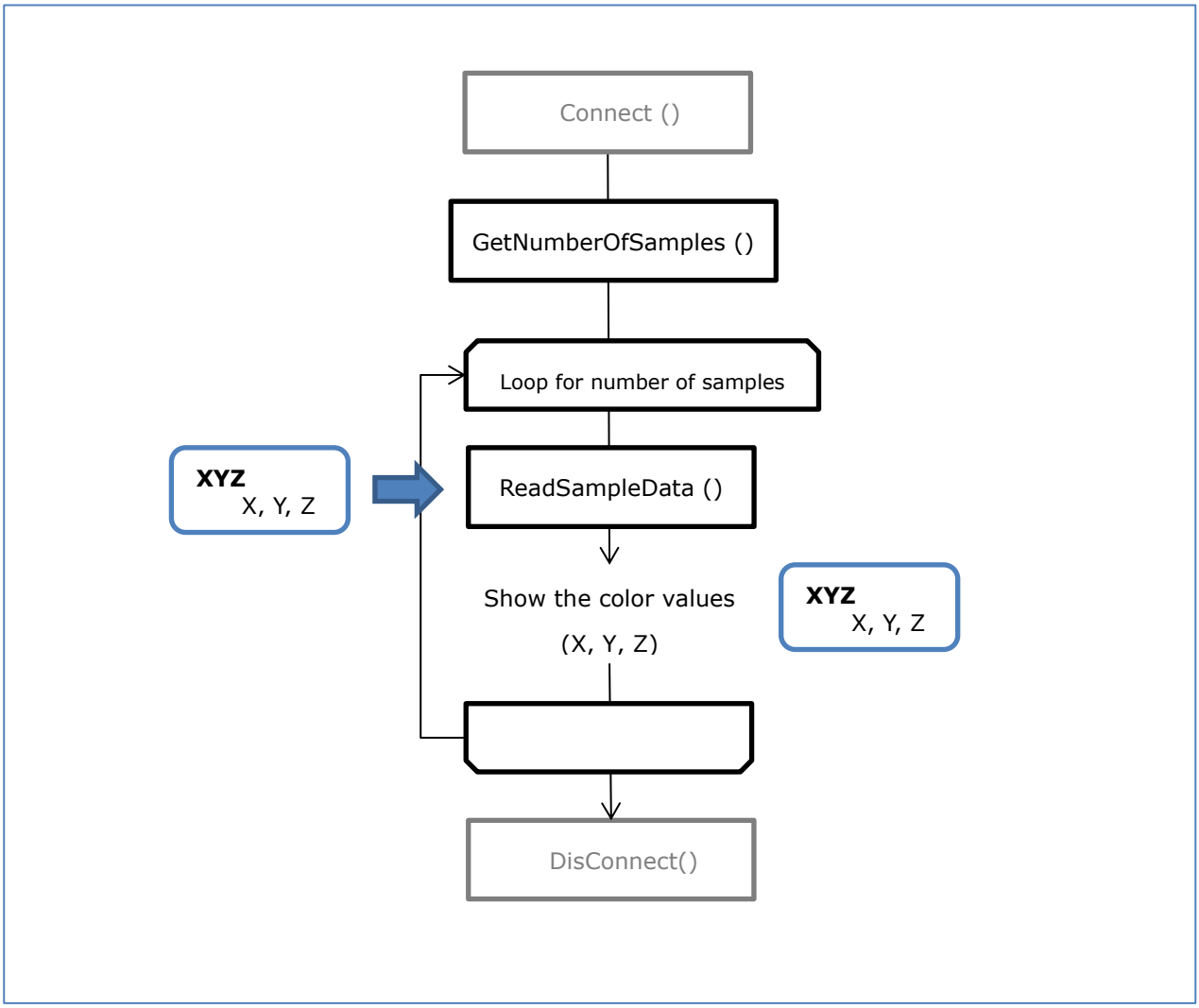
OneShotMeasurementSample.cs

```
using System;
using Konicaminolta;

namespace SampleProgram
{
    class OneShotMeasurementSample
    {
        public static void Execute()
        {

            //  Setup
            LightColorMISDK sdk = LightColorMISDK.GetInstance();
            ReturnMessage ret;


            //  Connect
            ret = sdk.Connect();
            if( ret.errorCode != ErrorDefine.KmSuccess )
            {
                Console.WriteLine("Error in Connect(): {0}", ret.errorCode);
                return;
            }

            //  Start measurement
            ret = sdk.Measure();
            if( ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in Measure(): {0}", ret.errorCode);
                return;
            }

            //  Polling status of measurement
            MeasStatus state;
            do
            {
                ret = sdk.PollingMeasurement(out state);
                if (ret.errorCode != ErrorDefine.KmSuccess)
                {
                    Console.WriteLine("Error in PollingMeasurement(): {0}", ret.errorCode);
                    return;
                }
            } while (state == MeasStatus.Measuring);

            //  Get measured value as ( X, Y, Z )
            XYZ xyzValue = new XYZ();
            ret = sdk.ReadLatestData(xyzValue);
            if( ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in ReadLatestData(): {0}", ret.errorCode);
                return;
            }
            else
            {
                // Display measurement values
                Console.WriteLine("Result");
                Console.WriteLine("X:{0}", xyzValue.X);
                Console.WriteLine("Y:{0}", xyzValue.Y);
```

```
            Console.WriteLine("Z:{0}", xyzValue.Z);
        }

        // Get measured Value as (Lv[cd/m2], x, y)
        Lvxy lvxyValue = new Lvxy(LuminanceUnit.cdm2);
        ret = sdk.ReadLatestData(lvxyValue);
        if (ret.errorCode != ErrorDefine.KmSuccess)
        {
            Console.WriteLine("Error in ReadLatestData(): {0}", ret.errorCode);
            return;
        }
        else
        {
            // Display measurement values
            Console.WriteLine("Result");
            Console.WriteLine("Lv:{0}", lvxyValue.Lv);
            Console.WriteLine(" x:{0}", lvxyValue.x);
            Console.WriteLine(" y:{0}", lvxyValue.y);
        }

        //  Disconnect
        ret = sdk.DisConnect(0);

    }
  }
}
```

2. Read the measurement data stored in the instrument

ID	Use case name		Overview
2	Obtain the measurement value stored in the instrument as a value in a certain color space.	1	Execute a measurement on the instrument without being connected to PC.
		2	Connect the instrument to the PC.
		3	Obtain the number of measurement values stored in the instrument.
		4	Specify the color space and obtain the measurement values.
		5	Execute step 4 repeatedly for the number of values obtained in step 3.



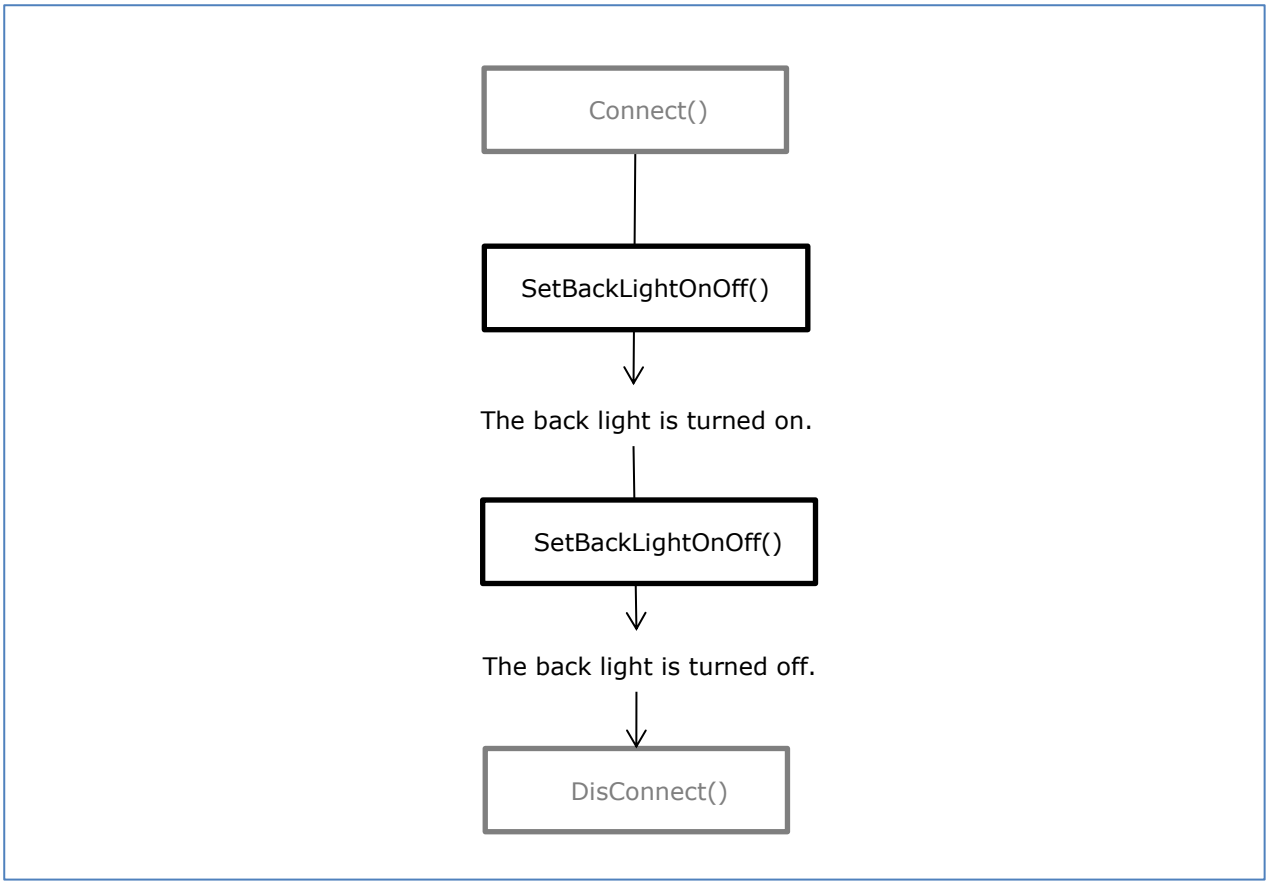
ReadSampleDataSample.cs

```
using System;
using Konicaminolta;

namespace SampleProgram
{
    class ReadSampleDataSample
    {
        public static void Execute()
        {

            //  Setup
            LightColorMISDK sdk = LightColorMISDK.GetInstance();
            ReturnMessage ret;

            //  Connect
            ret = sdk.Connect();
            if (ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in Connect(): {0}", ret.errorCode);
                return;
            }

            //
            int totalSampleCount;
            ret = sdk.GetNumberOfSampleData(out totalSampleCount);
            if (ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in Connect(): {0}", ret.errorCode);
                return;
            }

            //
            if (totalSampleCount == 0)
            {
                Console.WriteLine("No sample data");
                return;
            }

            //  Get Sample measured value as ( X, Y, Z )
            XYZ xyzValue = new XYZ();
            for (int i = 1; i <= totalSampleCount; i++)
            {
                ret = sdk.ReadSampleData(i, xyzValue);
                if (ret.errorCode != ErrorDefine.KmSuccess)
                {
                    Console.WriteLine("Error in ReadSampleData(): {0}", ret.errorCode);
                    return;
                }
                else
                {
                    // Display measurement values
                    Console.WriteLine("Result : DataNumber{0}", i);
                    Console.WriteLine("X:{0}", xyzValue.X);
                    Console.WriteLine("Y:{0}", xyzValue.Y);
                    Console.WriteLine("Z:{0}", xyzValue.Z);
                }
            }
```

```
            //  Disconnect
            ret = sdk.DisConnect(0);
        }
    }
}
```

3. Change the backlight setting

ID	Use case name		Overview
3	Change the backlight setting.	1	Turn on the backlight.
		2	Wait one second.
		3	Turn off the backlight.

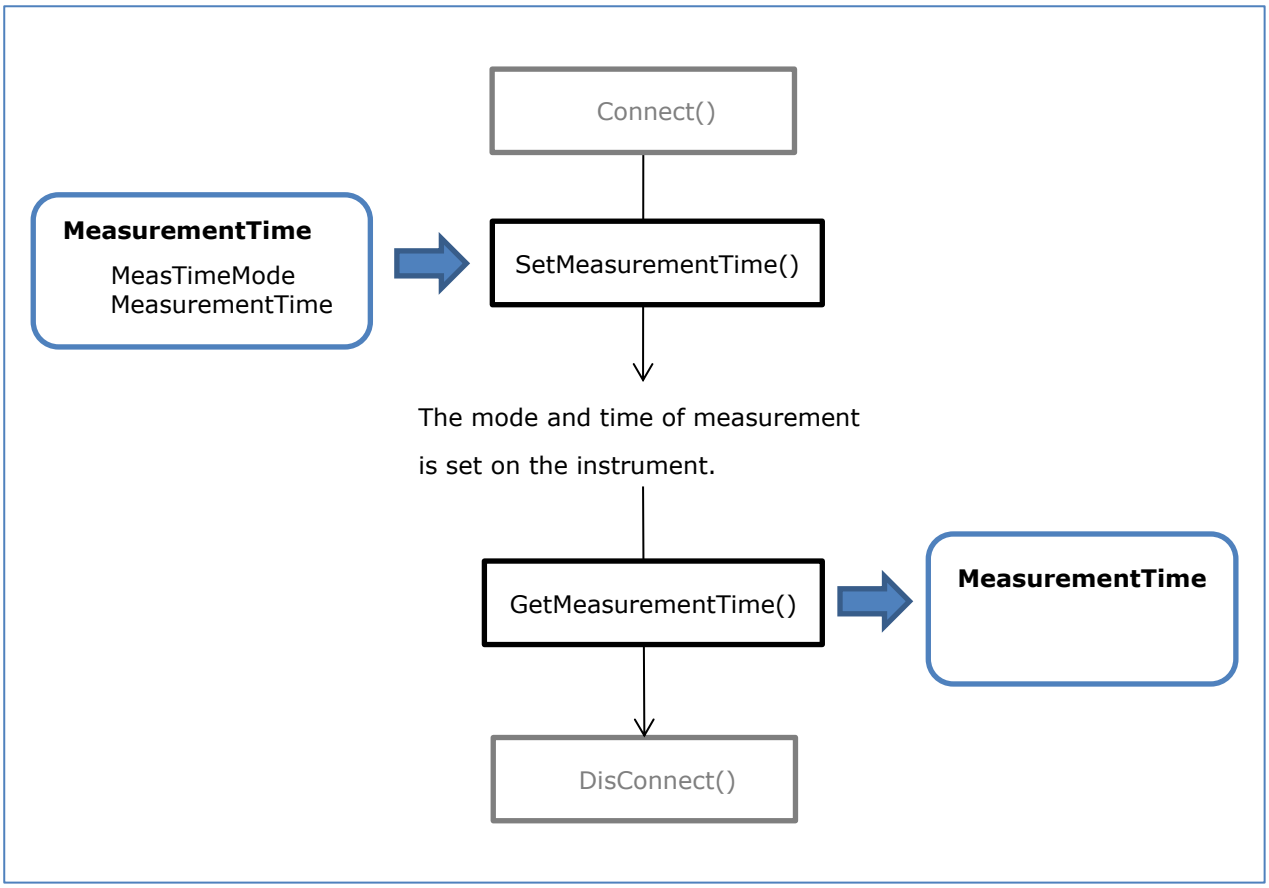


BackLightSettingSample.cs

```
using System;
using Konicaminolta;
using System.Threading;

namespace SampleProgram
{
    class BackLightSettingSample
    {
        public static void Execute()
        {

            //  Setup
            LightColorMISDK sdk = LightColorMISDK.GetInstance();
            ReturnMessage ret;

            //  Connect
            ret = sdk.Connect();
            if (ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in Connect(): {0}", ret.errorCode);
                return;
            }

            // Set the backlight on
            BackLightMode mode = BackLightMode.On;
            ret = sdk.SetBackLightOnOff(mode);
            if (ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in SetBackLightOnOff(): {0}", ret.errorCode);
                return;
            }

            Thread.Sleep(2000);

            // Set the backlight off
            mode = BackLightMode.Off;
            ret = sdk.SetBackLightOnOff(mode);
            if (ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in SetBackLightOnOff(): {0}", ret.errorCode);
                return;
            }

            // Disconnect
            ret = sdk.DisConnect();

        }
    }
}
```

4. Change the measurement time setting

ID	Use case name		Overview
4	Change the measurement time setting.	1	Create an instance of the MeasurementTime class and specify the synchronous/asynchronous setting.
		2	For synchronous, specify the synchronization frequency.
		3	Change the measurement time setting.
		4	Confirm the settings.



MeasurementTimeSettingSample.cs

```
using System;
using Konicaminolta;

namespace SampleProgram
{
    class MeasurementTimeSample
    {
        public static void Execute()
        {

            //  Setup
            LightColorMISDK sdk = LightColorMISDK.GetInstance();
            ReturnMessage ret;


            //  Connect
            ret = sdk.Connect();
            if (ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in Connect(): {0}", ret.errorCode);
                return;
            }

            //  Set the measurement time automatic
            MeasurementTime inputMeasurementTime = new MeasurementTime();
            inputMeasurementTime.MeasTimeMode = MeasTimeMode.Auto;
            ret = sdk.SetMeasurementTime(inputMeasurementTime);
            if (ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in SetMeasurementTime(): {0}", ret.errorCode);
                return;
            }

            MeasurementTime outputMeasurementTime;
            ret = sdk.GetMeasurementTime(out outputMeasurementTime);
            if (ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in GetMeasurementTime(): {0}", ret.errorCode);
                return;
            }
            else
            {
                // Display measurement time values
                Console.WriteLine("MeasTimeMode:{0}", inputMeasurementTime.MeasTimeMode);
            }

            //  Set a measurement time manually
            inputMeasurementTime.MeasTimeMode = MeasTimeMode.Manual;
            inputMeasurementTime.ManualMeasurementTime = 1.5;

            ret = sdk.SetMeasurementTime(inputMeasurementTime);
            if (ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in SetMeasurementTime(): {0}", ret.errorCode);
                return;
            }
            else
```

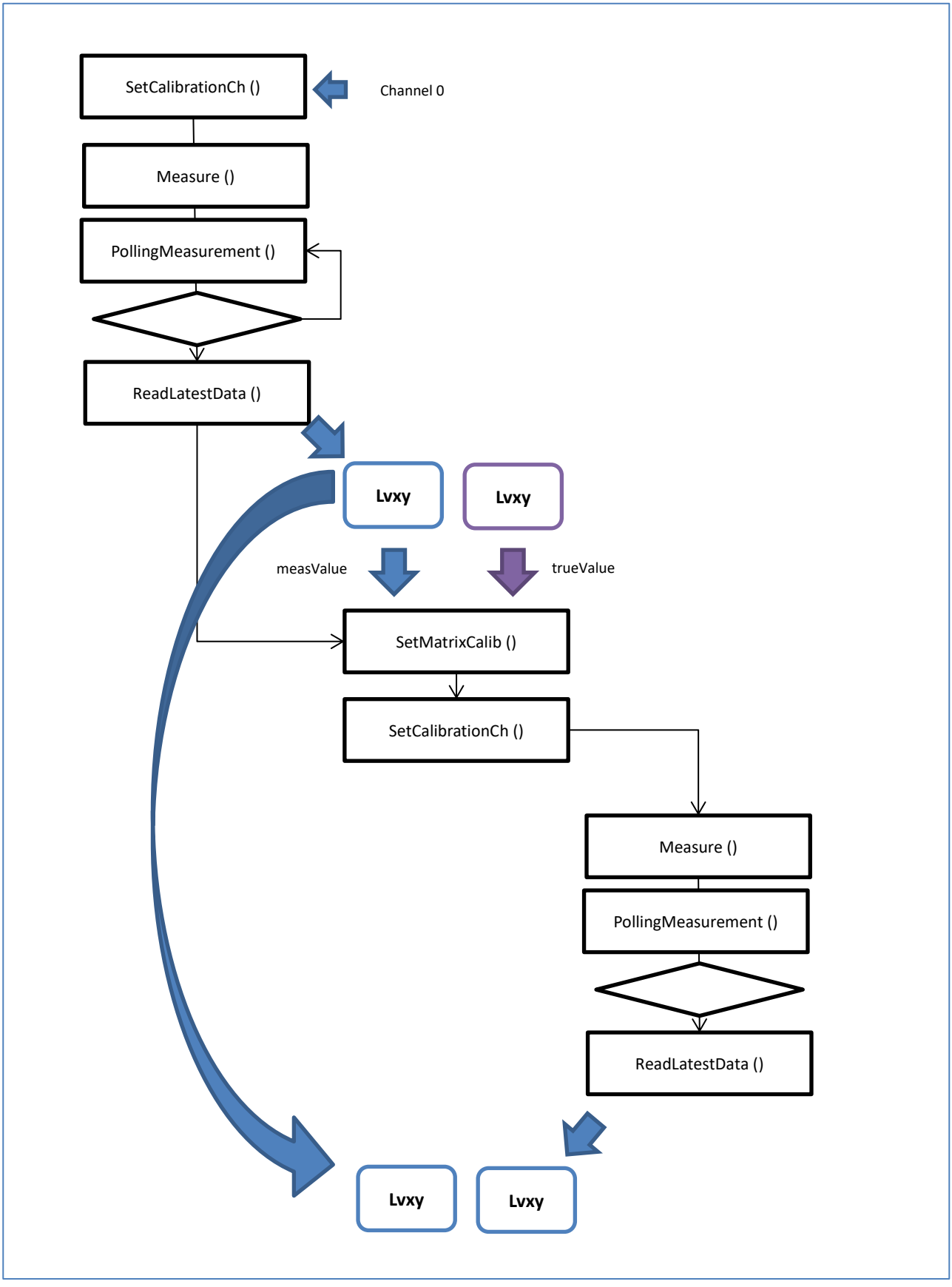
```
            {
                // Display measurement time values
                Console.WriteLine("");
                Console.WriteLine("MeasTimeMode:{0}", inputMeasurementTime.MeasTimeMode);
                Console.WriteLine("ManualMeasurementTime:{0}" ,
inputMeasurementTime.ManualMeasurementTime);
            }



            //  Disconnect
            ret = sdk.DisConnect(0);
        }
    }
}
```

5. Run single-point calibration

User calibration is a function that allows measurements to be taken with calibration coefficients applied to them by specifying a measurement value (actual measurement value for calibration) and the value that the measurement value should be (true value for calibration) as the calibration value set. For the API that will cause the user calibration coefficients to be set, first specify the user calibration type, the required calibration value set, and the calibration channel into which the calibration coefficients will be set. Then measurements can be taken with the calibration coefficients applied by specifying the calibration channel with another API.

ID	Use case name		Overview
6	Run single-point calibration based on a measurement value. Then obtain the data that was measured using this calibration value.	1	Set calibration channel to 0. (*1)
		2	Measure the white light source before user calibration.
		3	Obtain the measurement value before user calibration and set this as the actual measurement value for calibration
		4	Create the true value for calibration.
		5	Set the calibration type to single-point calibration and set the calibration value on the instrument.
		6	Specify the calibration channel setting for the instrument to the channel specified in step 4.
		7	After calibration, measure the same white light source as in step 1.
		8	Obtain the measurement value based on user calibration.
		9	Display the measurement values before and after user calibration.

*1. To avoid unintended double calibration, the measurement value before user calibration should be measured using calibration channel 0.


SetCalibrationCh ()
Channel 0
Measure ()
PollingMeasurement ()
ReadLatestData ()
Lvxy
Lvxy
measValue
trueValue
SetMatrixCalib ()
SetCalibrationCh ()
Measure ()
PollingMeasurement ()
ReadLatestData ()
Lvxy
Lvxy

SinglePointCalibrationSample.cs

```
using System;
using System.Collections.Generic;
using Konicaminolta;

namespace SampleProgram
{
    class SinglePointCalibrationSample
    {

        public static void Execute()
        {

            // Setup
            LightColorMISDK sdk = LightColorMISDK.GetInstance();
            ReturnMessage ret;

            // Connect
            ret = sdk.Connect();
            if (ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in Connect():{0}", ret.errorCode);
                return;
            }

            // Set calibration channel 0
            ret = sdk.SetCalibrationCh(0);
            if (ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in SetCalibrationCh():{0}", ret.errorCode);
                return;
            }

            // Start measurement (of a white point) before user calibration
            ret = sdk.Measure();
            if( ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in Measure(): {0}", ret.errorCode);
                return;
            }

            //  Polling status of measurement
            MeasStatus state;
            do
            {
                ret = sdk.PollingMeasurement(out state);
                if (ret.errorCode != ErrorDefine.KmSuccess)
                {
                    Console.WriteLine("Error in PollingMeasurement(): {0}", ret.errorCode);
                    return;
                }
            } while (state == MeasStatus.Measuring);

            // Read the value(Lv, x, y) before user calibration
            Lvxy beforeValue     = new Lvxy(LuminanceUnit.cdm2);
            ret = sdk.ReadLatestData(beforeValue);
            if( ret.errorCode != ErrorDefine.KmSuccess)
            {
```

```
            Console.WriteLine("Error in ReadLatestData(): {0}", ret.errorCode);
            return;
        }

        // Prepare measurement values for calibration
        List<Lvxy> measValues     = new List<Lvxy>();
        measValues.Add(beforeValue);
        List<Lvxy> trueValues     = new List<Lvxy>();
        trueValues.Add( new Lvxy(LuminanceUnit.cdm2, 11.0, 0.4, 0.4) );

        // Set calibration with ID
        CalibType type      = CalibType.OnePoint;
        string id           = "some_id";    // Set desired id
        int calibrationCh   = 1;
        ret = sdk.SetMatrixCalib(calibrationCh, measValues, trueValues, type, id);
        if (ret.errorCode != ErrorDefine.KmSuccess)
        {
            Console.WriteLine("Error in SetMatrixCalib():{0}", ret.errorCode);
            return;
        }

        //  Set calibration channel
        ret = sdk.SetCalibrationCh(calibrationCh);
        if (ret.errorCode != ErrorDefine.KmSuccess)
        {
            Console.WriteLine("Error in SetCalibrationCh():{0}", ret.errorCode);
            return;
        }

        //  Start measurement (of a white point) after calibration
        ret = sdk.Measure();
        if( ret.errorCode != ErrorDefine.KmSuccess)
        {
            Console.WriteLine("Error in Measure(): {0}", ret.errorCode);
            return;
        }

        //  Polling status of measurement
        do
        {
            ret = sdk.PollingMeasurement(out state);
            if( ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in PollingMeasurement(): {0}", ret.errorCode);
                return;
            }
        } while( state == MeasStatus.Measuring );

        // Read the value(Lv, x, y) after calibration
        Lvxy afterValue = new Lvxy(LuminanceUnit.cdm2);
        ret = sdk.ReadLatestData(afterValue);
        if( ret.errorCode != ErrorDefine.KmSuccess)
        {
            Console.WriteLine("Error in ReadLatestData(): {0}", ret.errorCode);
            return;
        }

        //
        Console.WriteLine("Before user calibration");
```

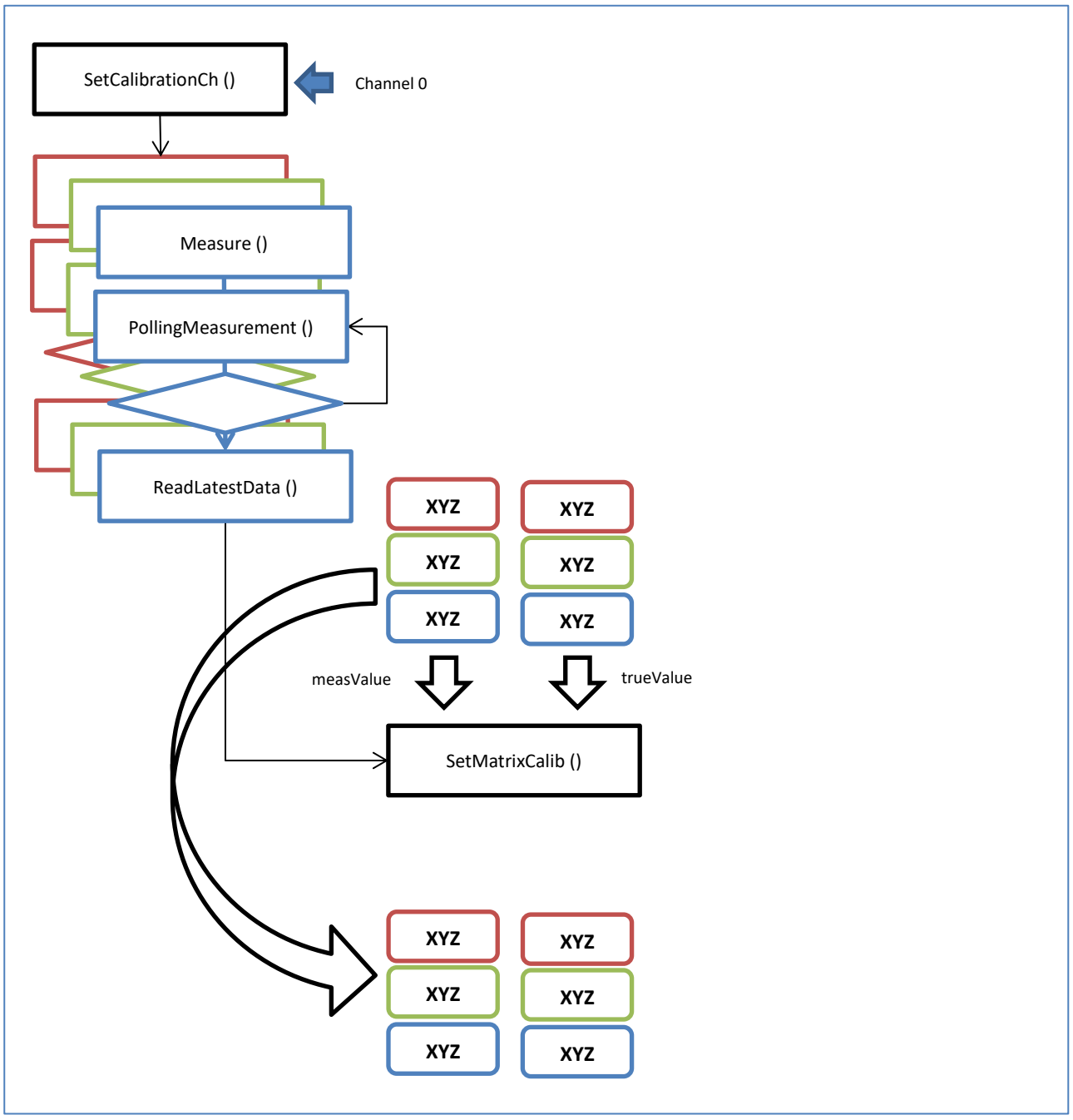
```
            foreach(var beforeCalibValue in beforeValue.ColorSpaceValue){
                Console.WriteLine("{0} : {1}", beforeCalibValue.Key,
beforeCalibValue.Value);
            }
            Console.WriteLine("After user calibration");
            foreach(var afterCalibValue in afterValue.ColorSpaceValue){
                Console.WriteLine("{0} : {1}", afterCalibValue.Key, afterCalibValue.Value);
            }

            //  Disconnect
            ret = sdk.DisConnect(0);
        }
    }
}
```

6. Run RGB calibration

Similar to single-point calibration, RGB calibration is a function that allows measurements to be taken with calibration coefficients applied to them by specifying a measurement value (actual measurement value for calibration) and the value that the measurement value should be (true value for calibration) as the calibration value set. What differs from single-point calibration is that three light sources (red, green, blue) are measured and three calibration value sets (actual measurement value for calibration and true value for calibration for each light source) must be prepared.

ID	Use case name		Overview
6	Run single-point calibration based on a measurement value. Then obtain the data that was measured using this calibration value.	1	Set calibration channel to 0 (*1)
		R1	Measure the red light source (R) before user calibration
		R2	Obtain the measurement value before user calibration and set this as the actual measurement value for calibration "R(X,Y,Z)"
		G1	Measure the green light source (G) before user calibration
		G2	Obtain the measurement value before user calibration and set this as the actual measurement value for calibration "G(X,Y,Z)"
		B1	Measure a blue light source (B) before user calibration
		B2	Obtain the measurement value before user calibration and set this as the actual measurement value for calibration "B(X,Y,Z)"
		2	Create the true values for calibration "R(X,Y,Z), G(X,Y,Z), B(X,Y,Z)"
		3	Set the calibration type to RGB calibration and set the calibration values on the instrument
		4	Display the measurement values before and the true values for calibration

*1. To avoid unintended double calibration, obtaining the measurement value before calibration should be peformed with calibration channel 0.


SetCalibrationCh ()
Channel 0
Measure ()
PollingMeasurement ()
ReadLatestData ()
XYZ
XYZ
XYZ
XYZ
XYZ
XYZ
measValue
trueValue
SetMatrixCalib ()
XYZ
XYZ
XYZ
XYZ
XYZ
XYZ

RGBCalibrationSample.cs

```
using System;
using System.Collections.Generic;
using Konicaminolta;

namespace SampleProgram
{
    class RGBCalibrationSample
    {
        public static void Execute()
        {

            // Setup
            LightColorMISDK sdk = LightColorMISDK.GetInstance();
            ReturnMessage ret;

            //  Connect
            ret = sdk.Connect();
            if (ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in Connect():{0}", ret.errorCode);
                return;
            }

            // Set calibration channel 0
            ret = sdk.SetCalibrationCh(0);
            if (ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in SetCalibrationCh():{0}", ret.errorCode);
                return;
            }

            //  Start measurement (of a red light)
            Console.WriteLine("Press any key to measure a red light");
            Console.ReadKey();
            ret = sdk.Measure();
            if( ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in Measure(): {0}", ret.errorCode);
                return;
            }

            //  Polling status of measurement
            MeasStatus state;
            do
            {
                ret = sdk.PollingMeasurement(out state);
                if (ret.errorCode != ErrorDefine.KmSuccess)
                {
                    Console.WriteLine("Error in PollingMeasurement(): {0}", ret.errorCode);
                    return;
                }
            } while (state == MeasStatus.Measuring);

            //  Get measured value as ( X, Y, Z )
            XYZ measRedValue = new XYZ();
            ret = sdk.ReadLatestData(measRedValue);
            if( ret.errorCode != ErrorDefine.KmSuccess)
```

```
            {
                Console.WriteLine("Error in ReadLatestData(): {0}", ret.errorCode);
                return;
            }

            //  Start measurement (of a green light)
            Console.WriteLine("Press any key to measure a green light");
            Console.ReadKey();
            ret = sdk.Measure();
            if( ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in Measure(): {0}", ret.errorCode);
                return;
            }

            //  Polling status of measurement
            do {
                ret = sdk.PollingMeasurement(out state);
                if( ret.errorCode != ErrorDefine.KmSuccess)
                {
                    Console.WriteLine("Error in PollingMeasurement(): {0}", ret.errorCode);
                    return;
                }
            } while( state == MeasStatus.Measuring );

            //  Get measured value as ( X, Y, Z )
            XYZ measGreenValue = new XYZ();
            ret = sdk.ReadLatestData(measGreenValue);
            if( ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in ReadLatestData(): {0}", ret.errorCode);
                return;
            }

            //  Start measurement (of a blue light)
            Console.WriteLine("Press any key to measure a blue light");
            Console.ReadKey();
            ret = sdk.Measure();
            if( ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in Measure(): {0}", ret.errorCode);
                return;
            }

            //  Polling status of measurement
            do
            {
                ret = sdk.PollingMeasurement(out state);
                if (ret.errorCode != ErrorDefine.KmSuccess)
                {
                    Console.WriteLine("Error in PollingMeasurement(): {0}", ret.errorCode);
                    return;
                }
            } while (state == MeasStatus.Measuring);


            //  Get measured value as ( X, Y, Z )
            XYZ measBlueValue = new XYZ();
            ret = sdk.ReadLatestData(measBlueValue);
```

```
            if( ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in ReadLatestData(): {0}", ret.errorCode);
                return;
            }

            List<XYZ> measValues = new List<XYZ>();
            measValues.Add(measRedValue);
            measValues.Add(measGreenValue);
            measValues.Add(measBlueValue);

            XYZ trueRedValue = new XYZ();
            trueRedValue.X = 800;
            trueRedValue.Y = 400;
            trueRedValue.Z = 300;
            XYZ trueGreenValue = new XYZ();
            trueGreenValue.X = 600;
            trueGreenValue.Y = 1000;
            trueGreenValue.Z = 400;
            XYZ trueBlueValue = new XYZ();
            trueBlueValue.X = 500;
            trueBlueValue.Y = 600;
            trueBlueValue.Z = 1000;
            List<XYZ> trueValues = new List<XYZ>();
            trueValues.Add(trueRedValue);
            trueValues.Add(trueGreenValue);
            trueValues.Add(trueBlueValue);

            // Set calibration coefs
            string id      = "some_id";    // Set desired id
            CalibType type = CalibType.RGB;
            int calibrationCh = 1;
            ret = sdk.SetMatrixCalib(calibrationCh, measValues, trueValues, type, id);
            if (ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in SetMatrixCalib():{0}", ret.errorCode);
                return;
            }

            // Get calibration coefs
            List<MeasurementData> outputMeasValues = new List<MeasurementData>();
            List<MeasurementData> outputTrueValues = new List<MeasurementData>();
            UserCalibData calibCoefs = new UserCalibData();

            ret = sdk.GetCalibData(calibrationCh, out outputMeasValues, out outputTrueValues,
out calibCoefs);
            if (ret.errorCode != ErrorDefine.KmSuccess)
            {
                Console.WriteLine("Error in GetCalibData():{0}", ret.errorCode);
                return;
            }

            //
            Console.WriteLine("Before user calibration (Red)");
            foreach(var _measRedValue in outputMeasValues[0].ColorSpaceValue){
                Console.WriteLine("{0} : {1}", _measRedValue.Key, _measRedValue.Value);
            }

            Console.WriteLine("After user calibration (Red)");
```

```
            foreach(var _trueRedValue in outputTrueValues[0].ColorSpaceValue){
                Console.WriteLine("{0} : {1}", _trueRedValue.Key, _trueRedValue.Value);
            }

            Console.WriteLine("Before user calibration (Green)");
            foreach(var _measGreenValue in outputMeasValues[1].ColorSpaceValue){
                Console.WriteLine("{0} : {1}", _measGreenValue.Key, _measGreenValue.Value);
            }

            Console.WriteLine("After user calibration (Green)");
            foreach(var _trueGreenValue in outputTrueValues[1].ColorSpaceValue){
                Console.WriteLine("{0} : {1}", _trueGreenValue.Key, _trueGreenValue.Value);
            }

            Console.WriteLine("Before user calibration (Blue)");
            foreach(var _measBlueValue in outputMeasValues[2].ColorSpaceValue){
                Console.WriteLine("{0} : {1}", _measBlueValue.Key, _measBlueValue.Value);
            }

            Console.WriteLine("After user calibration (Blue)");
            foreach(var _trueBlueValue in outputTrueValues[2].ColorSpaceValue){
                Console.WriteLine("{0} : {1}", _trueBlueValue.Key, _trueBlueValue.Value);
            }

            //  Disconnect
            ret = sdk.DisConnect(0);

        }
    }
}
```

3.4 Supported API List (CS-150/CS-160)

Group	API Name	API Description
Communication	Connect()	Connects to the instrument.
Communication	DisConnect()	Disconnects from the instrument.
Communication	GetDeviceList()	Obtains information about the instruments connected to the PC.
Measurement	Measure()	Executes one sample measurement.
Measurement	PollingMeasurement()	Checks the measurement status.
Measurement	CancelMeasurement()	Cancels the measurement.
Current Measurement Data	ReadLatestData()	Obtains the measurement data for the latest measurement.
Current Measurement Data	ReadDisplayValue()	Obtains the displayed values.
Measurement Data (Stored in Device)	GetNumberOfSampleData()	Obtains the number of measurement data stored in the instrument.
Measurement Data (Stored in Device)	ReadSampleData()	Obtains the stored measurement data from the instrument.
Measurement Data (Stored in Device)	DeleteSampleData()	Deletes the stored measurement data.
Target Value	SetTargetCh()	Sets the target channel that will be used.
Target Value	GetTargetCh()	Obtains the currently specified target channel.
Target Value	ReadTargetData()	Obtains the target data from the instrument.
Target Value	DeleteTargetData()	Deletes the target data.
Target Value	WriteTargetData()	Writes the target data that will be used to the instrument.
Measurement Time	SetMeasurementTime()	Sets the measurement time.
Measurement Time	GetMeasurementTime()	Obtains the measurement time.
Synchronization	SetSyncMode()	Sets measurement synchronization setting.
Synchronization	GetSyncMode()	Obtains the measurement synchronization setting.
Peak/Valley	SetPeakValley()	Sets peak/valley setting.

Peak/Valley	GetPeakValley()	Obtains the peak/valley setting.
Lens Settings	SetCloseUpLens()	Sets the close-up lens setting.
Lens Settings	GetCloseUpLens()	Obtains the close-up lens setting.
User Calibration Channel	SetCalibrationCh()	Sets the user calibration channel that will be used.
User Calibration Channel	GetCalibrationCh()	Obtains the currently specified user calibration channel.
User Calibration	SetMatrixCalib()	Sets the user calibration type and measured/true values.
User Calibration	GetCalibData()	Obtains the user calibration data (measured/true values and coefficients).
User Calibration	DeleteCalibData()	Deletes the user calibration data.
Power Settings	SetAutoPowerOff()	Sets the auto power off setting.
Power Settings	GetAutoPowerOff()	Obtains the auto power off setting.
Backlight	SetBackLightOnOff()	Sets the backlight setting.
Backlight	GetBackLightOnOff()	Obtains the backlight setting.
Backlight	SetBackLightLevel()	Sets the backlight brightness setting.
Backlight	GetBackLightLevel()	Obtains the backlight brightness setting.
Display Digits	SetColorDispDigit()	Sets the color display digits setting.
Display Digits	GetColorDispDigit()	Obtains the color display digits setting.
Display Type (Absolute Value/Difference/Ratio)	SetDisplayType()	Sets the display type setting.
Display Type (Absolute Value/Difference/Ratio)	GetDisplayType()	Obtains the display type setting.
Language/Date/Time	SetDisplayLanguage()	Sets the display language setting.
Language/Date/Time	GetDisplayLanguage()	Obtains the display language setting.
Language/Date/Time	SetDateTime()	Sets the date/time setting.
Language/Date/Time	GetDateTime()	Obtains the date/time setting.
Language/Date/Time	SetDateFormat()	Sets the date/time format.
Language/Date/Time	GetDateFormat()	Obtains the date/time format.
Device Color Mode	SetColorModeDisplayOnOff()	Sets the on/off setting for each color mode.
Device Color Mode	GetColorModeDisplayOnOff()	Obtains the on/off setting for each color mode.

Device Color Mode	SetColorMode()	Sets the color mode setting.
Device Color Mode	GetColorMode()	Obtains the color mode setting.
Data Save Mode	SetDataSaveMode()	Sets the data save mode setting.
Data Save Mode	GetDataSaveMode()	Obtains the data save mode setting.
Measurement Calibration Settings	SetPeriodicCalibNotify()	Sets the periodic calibration message display setting.
Measurement Calibration Settings	GetPeriodicCalibNotify()	Obtains the periodic calibration message display setting.
Button Settings	SetToggleOnOff()	Sets the toggle setting for the instrument measurement button.
Button Settings	GetToggleOnOff()	Obtains the toggle setting for the instrument measurement button.
Button Settings	SetTriggerOnOff()	Sets the enabled/disabled status for the instrument measurement button.
Button Settings	GetTriggerOnOff()	Obtains the enabled/disabled status for the instrument measurement button.
Device/SDK Information	GetDeviceInfo()	Obtains the instrument information.
Device/SDK Information	GetSDKVersion()	Obtains the SDK version.
Measurement Unit Settings	SetLuminanceUnit()	Sets the luminance unit setting.
Measurement Unit Settings	GetLuminanceUnit()	Obtains the luminance unit setting.

3.5 Supported API List (LS-150/LS-160)

Group	API Name	API Description
Communication	Connect()	Connects to the instrument.
Communication	DisConnect()	Disconnects from the instrument.
Communication	GetDeviceList()	Obtains information about the instruments connected to the PC.
Measurement	Measure()	Executes one sample measurement.
Measurement	PollingMeasurement()	Checks the measurement status.
Measurement	CancelMeasurement()	Cancels the measurement.
Current Measurement Data	ReadLatestData()	Obtains the measurement data for the latest measurement.
Current Measurement Data	ReadDisplayValue()	Obtains the displayed value.
Measurement Data (Stored in Device)	GetNumberOfSampleData()	Obtains the number of measurement data stored in the instrument.
Measurement Data (Stored in Device)	ReadSampleData()	Obtains the stored measurement data from the instrument.
Measurement Data (Stored in Device)	DeleteSampleData()	Deletes the stored measurement data.
Target Value	SetTargetCh()	Sets the target channel that will be used.
Target Value	GetTargetCh()	Obtains the currently specified target channel.
Target Value	ReadTargetData()	Obtains the target data from the instrument.
Target Value	DeleteTargetData()	Deletes the target data.
Target Value	WriteTargetData()	Writes the target data that will be used to the instrument.
Measurement Time	SetMeasurementTime()	Sets the measurement time.
Measurement Time	GetMeasurementTime()	Obtains the measurement time.
Synchronization	SetSyncMode()	Sets the measurement synchronization setting.
Synchronization	GetSyncMode()	Obtains the measurement synchronization setting.
Peak/Valley	SetPeakValley()	Sets the peak/valley setting.

Peak/Valley	GetPeakValley()	Obtains the peak/valley setting.
Lens Settings	SetCloseUpLens()	Sets the close-up lens setting.
Lens Settings	GetCloseUpLens()	Obtains the close-up lens setting.
User Calibration Channel	SetCalibrationCh()	Sets the user calibration channel that will be used.
User Calibration Channel	GetCalibrationCh()	Obtains the currently specified user calibration channel.
User Calibration	SetMatrixCalib()	Sets the user calibration type and measured/true values.
User Calibration	GetCalibData()	Obtains the user calibration data (measured/true values and coefficients.
User Calibration	DeleteCalibData()	Deletes the user calibration data.
CCF	SetCCF()	Specifies the CCF (color correction factor) setting.
CCF	GetCCF()	Obtains the CCF (color correction factor) setting.
Power Settings	SetAutoPowerOff()	Sets the auto power off setting.
Power Settings	GetAutoPowerOff()	Obtains the auto power off setting.
Backlight	SetBackLightOnOff()	Sets the backlight setting.
Backlight	GetBackLightOnOff()	Obtains the backlight setting.
Backlight	SetBackLightLevel()	Sets the backlight brightness setting.
Backlight	GetBackLightLevel()	Obtains the backlight brightness setting.
Display Type (Absolute Value/Difference/Ratio)	SetDisplayType()	Sets the display type setting.
Display Type (Absolute Value/Difference/Ratio)	GetDisplayType()	Obtains the display type setting.
Language/Date/Time	SetDisplayLanguage()	Sets the display language setting.
Language/Date/Time	GetDisplayLanguage()	Obtains the display language setting.
Language/Date/Time	SetDateTime()	Sets the date/time setting.
Language/Date/Time	GetDateTime()	Obtains the date/time setting.
Language/Date/Time	SetDateFormat()	Sets the date/time format.
Language/Date/Time	GetDateFormat()	Obtains the date/time format.
Data Save Mode	SetDataSaveMode()	Sets the data save mode setting.

Data Save Mode	GetDataSaveMode()	Obtains the data save mode setting.
Measurement Calibration Settings	SetPeriodicCalibNotify()	Sets the periodic calibration message display setting.
Measurement Calibration Settings	GetPeriodicCalibNotify()	Obtains the periodic calibration message display setting.
Button Settings	SetToggleOnOff()	Sets the toggle setting for the instrument measurement button.
Button Settings	GetToggleOnOff()	Obtains the toggle setting for the instrument measurement button.
Button Settings	SetTriggerOnOff()	Sets the enabled/disabled status for the instrument measurement button.
Button Settings	GetTriggerOnOff()	Obtains the enabled/disabled status for the instrument measurement button.
Device/SDK Information	GetDeviceInfo()	Obtains the instrument information.
Device/SDK Information	GetSDKVersion()	Obtains the SDK version.
Measurement Unit Settings	SetLuminanceUnit()	Sets the luminance unit setting.
Measurement Unit Settings	GetLuminanceUnit()	Obtains the luminance unit setting.

3.6 Color Space

	XYZ	Lvxy	Lvudvd	LvTcpDuv	LvDwPe	Lv
CS-150	OK	OK	OK	OK	OK	-
CS-160	OK	OK	OK	OK	OK	-
LS-150	-	-	-	-	-	OK
LS-160	-	-	-	-	-	OK

4. API Specification

4.1 API List

Group	API Name	API Description
Communication	Connect()	Connects to the instrument.
Communication	DisConnect()	Disconnects from the instrument.
Communication	GetDeviceList()	Obtains information about the instruments connected to the PC.
Measurement	Measure()	Executes one sample measurement.
Measurement	PollingMeasurement()	Checks the measurement status.
Measurement	CancelMeasurement()	Cancels the measurement.
Current Measurement Data	ReadLatestData()	Obtains the measurement data for the latest measurement.
Current Measurement Data	ReadDisplayValue()	Obtains the displayed values.
Measurement Data (Stored in Device)	GetNumberOfSampleData()	Obtains the number of measurement data stored in the instrument.
Measurement Data (Stored in Device)	ReadSampleData()	Obtains the stored measurement data from the instrument.
Measurement Data (Stored in Device)	DeleteSampleData()	Deletes the stored measurement data.
Target Value	SetTargetCh()	Sets the target channel that will be used.
Target Value	GetTargetCh()	Obtains the currently specified target channel.
Target Value	ReadTargetData()	Obtains the target data from the instrument.
Target Value	DeleteTargetData()	Deletes the target data.
Target Value	WriteTargetData()	Writes the target data that will be used to the instrument.
Measurement Time	SetMeasurementTime()	Sets the measurement time.
Measurement Time	GetMeasurementTime()	Obtains the measurement time.
Synchronization	SetSyncMode()	Sets themeasurement synchronization setting.
Synchronization	GetSyncMode()	Obtains the measurement synchronization

		setting.
Peak/Valley	SetPeakValley()	Sets the peak/valley setting.
Peak/Valley	GetPeakValley()	Obtains the peak/valley setting.
Lens Settings	SetCloseUpLens()	Sets the close-up lens setting.
Lens Settings	GetCloseUpLens()	Obtains the close-up lens setting.
User Calibration Channel	SetCalibrationCh()	Sets the user calibration channel that will be used.
User Calibration Channel	GetCalibrationCh()	Obtains the currently specified user calibration channel.
User Calibration	SetMatrixCalib()	Sets the user calibration type and measured/true values.
User Calibration	GetCalibData()	Obtains the user calibration data (measured/true values and coefficients.
User Calibration	DeleteCalibData()	Deletes the user calibration data.
CCF	SetCCF()	Specifies the CCF (color correction factor) setting.
CCF	GetCCF()	Obtains the CCF (color correction factor) setting.
Power Settings	SetAutoPowerOff()	Sets the auto power off setting.
Power Settings	GetAutoPowerOff()	Obtains the auto power off setting.
Backlight	SetBackLightOnOff()	Sets the backlight setting.
Backlight	GetBackLightOnOff()	Obtains the backlight setting.
Backlight	SetBackLightLevel()	Sets the backlight brightness setting.
Backlight	GetBackLightLevel()	Obtains the backlight brightness setting.
Display Digits	SetColorDispDigit()	Sets the color display digits setting.
Display Digits	GetColorDispDigit()	Obtains the color display digits setting.
Display Type (Absolute Value/Difference/Ratio)	SetDisplayType()	Sets the display type setting.
Display Type (Absolute Value/Difference/Ratio)	GetDisplayType()	Obtains the display type setting.
Language/Date/Time	SetDisplayLanguage()	Sets the display language setting.
Language/Date/Time	GetDisplayLanguage()	Obtains the display language setting.
Language/Date/Time	SetDateTime()	Sets the date/time setting.

Language/Date/Time	GetDateTime()	Obtains the date/time setting.
Language/Date/Time	SetDateFormat()	Sets the date/time format.
Language/Date/Time	GetDateFormat()	Obtains the date/time format.
Device Color Mode	SetColorModeDisplayOnOff()	Sets the on/off setting for each color mode.
Device Color Mode	GetColorModeDisplayOnOff()	Obtains the on/off setting for each color mode.
Device Color Mode	SetColorMode()	Sets the color mode setting.
Device Color Mode	GetColorMode()	Obtains the color mode setting.
Data Save Mode	SetDataSaveMode()	Sets the data save mode setting.
Data Save Mode	GetDataSaveMode()	Obtains the data save mode setting.
Measurement Calibration Settings	SetPeriodicCalibNotify()	Sets the periodic calibration message display setting.
Measurement Calibration Settings	GetPeriodicCalibNotify()	Obtains the periodic calibration message display setting.
Button Settings	SetToggleOnOff()	Sets the toggle setting for the instrument measurement button.
Button Settings	GetToggleOnOff()	Obtains the toggle setting for the instrument measurement button.
Button Settings	SetTriggerOnOff()	Sets the enabled/disabled status for the instrument measurement button.
Button Settings	GetTriggerOnOff()	Obtains the enabled/disabled status for the instrument measurement button.
Device/SDK Information	GetDeviceInfo()	Obtains the instrument information.
Device/SDK Information	GetSDKVersion()	Obtains the SDK version.
Measurement Unit Settings	SetLuminanceUnit()	Sets the luminance unit setting.
Measurement Unit Settings	GetLuminanceUnit()	Obtains the luminance unit setting.

4.2 API Format

API specification format

The APIs are described using the following format.

Function:

Describes the processing that can be executed by the function.

Format:

Describes the format of the function.

Argument:

Describes the function arguments.

Return Value:

Describes the return values that may be returned when the function is used.

There are three types of return values.

Type	
Success	Returned when the processing was successful.
Error	Returned when the processing failed.
Warning	Returned when the processing was successful, but there is information that must be provided to the user.

Explanation:

Describes necessary information and precautions when using the function.

Class: ReturnMessage

All APIs return the following return value class.

Function:

The class used as the return value in the APIs.

Format:

```
class ReturnMessage
{
    Int32           errorCode;
    List<string>    errorMessage;
}
```

Variable:

Variable	Explanation
errorCode	Error code
errorMessage	Error Message (English)

Explanation:

The error code so the error can be broadly understood is stored in errorCode.

Details about the error code are stored in errorMessage.

Color Space Class

With the LC-MISDK, measurement values can be obtained in the desired color space and target values can be set in the desired color space using the APIs. To obtain or set a measurement value in certain color spaces, create instances of the color space classes and specify these as the API arguments. * The color spaces that can be used differ by the model of instrument.

For example, to obtain a measurement value as XYZ, create an instance of the XYZ class and specify it as an argument for the obtain measurement value API.

```
XYZ value = new XYZ();
sdk.ReadLatestData(value);
```

The X, Y, Z color values contained in the XYZ measurement value can be accessed as follows.

```
Console.WriteLine("X:{0}", value.X);
Console.WriteLine("Y:{0}", value.Y);
Console.WriteLine("Z:{0}", value.Z);
```

Obtain a target value in Lvxy as follows. With Lvxy, you can also specify the luminance unit in the constructor.

```
Lvxy lvxy = new Lvxy(LuminanceUnit.cdm2);

value.Lv = 1.0;
value.x  = 0.1;
value.y  = 0.1;

sdk.WriteTargetData(lvxyValue);
```

For detailed usage, see 5.2 Color Space Class

4.3 Communication

Connect()

Function:

Connects to the instrument that is connected to the specified virtual COM port.

Format:

ReturnMessage **Connect**(Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	Communication with the instrument connected to the specified virtual COM port was achieved.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErConnectFailed	Error	Failed to connect to the instrument.

Explanation:

If 0 is specified for comPort when using Connect() (the same as when no argument is specified), GetDeviceList() is automatically called and communication is established with the instrument with the lowest virtual COM port number. In such case, 0 should be specified for comPort in the other APIs to use the virtual COM port number of the instrument that was connected using Connect().

DisConnect()

Function:

Ends communication with the instrument connected to the specified virtual COM port.

Format:

ReturnMessage **DisConnect**(Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	Communication was ended with the instrument connected to the specified virtual COM port.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErDisConnectFailed	Error	Disconnection failed.

Explanation:

If there was an error, turn off the power to the instrument and disconnect it.

GetDeviceList()

Function:

Obtains the instrument information list (virtual COM port number, model name, serial number) for instruments connected to the PC(*1).

Format:

ReturnMessage **GetDeviceList**(out Dictionary<Int32, string> deviceList)

Argument:

Argument	I/O	Explanation
deviceList	O	List of instruments connected to the PC. The Dictionary key is the virtual COM port number. The value is the model name (serial number).

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The instrument list was obtained.
ErGetFailed	Error	The instrument list could not be obtained.

Explanation:

For example, if a CS-150 with the serial number 12345678 is connected to COM2, 2 is stored as the deviceList key and "CS-150(12345678)" is stored in value.

*1. "Connected to the PC" means instruments that are connected to the PC and recognized by the operating system, regardless of whether or not they are connected to the SDK.

4.4 Measurement

Measure()

Function:

Executes one measurement.

Format:

ReturnMessage **Measure**(Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The instrument started the measurement.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErMeasurementFailed	Error	Measurement failed.

Explanation:

Enum: MeasStatus

Value definitions that represent the measurement status

MeasStatus	**Definition value**	**Explanation**
Idling	0	Completed
Measuring	1	Measuring

PollingMeasurement()

Function:

Obtains the measurement status.

Format:

ReturnMessage **PollingMeasurement**(out MeasStatus measStatus, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
measStatus	O	Measurement status
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The measurement status was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErGetFailed	Error	Failed to obtain the measurement status.

Explanation:

CancelMeasurement()

Function:

Cancels the measurement.

Format:

ReturnMessage **CancelMeasurement**(Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The measurement was canceled.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCancelFailed	Error	Cancellation failed.

Explanation:

- If this function is executed when a measurement is not being executed, "KmSuccess" is returned as the return value.
- If CancelMeasurement is executed, the measurement that was being executed up to that point cannot be restarted.

4.5 Current Measurement Data

ReadLatestData()

Function:

Obtains the measurement data for the latest measurement.

Format:

ReturnMessage **ReadLatestData**("Desired color space class" measurementData , Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
measurementData	I/O	Measured data
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The measurement data were obtained.
ErNoConnect	Error	No instrument is connected to the specified COM port.
ErInvalidParameter	Error	Invalid parameter.
ErNoData	Error	No specified data. *1
ErReadFailed	Error	Failed to obtain the measurement data.
ErCalcFailed	Error	Calculation failed. *2
ErOutOfRangeValue	Error	Calculation result is out of the chromaticity or luminance range. *3

Explanation:

・Specify the variable of the color space class you want to use from the available color spaces as an argument.

・ The color value data that can be obtained is only for the latest measurement data.

*1. In this case, the calculated values are FLT_MAX.

*2. In this case, the calculated values are FLT_MIN.

*3. In this case, the calculated values are the values out of the chromaticity or luminance range.

ReadDisplayValue()

Function:

Obtains the display value.

Format:

ReturnMessage **ReadDisplayValue**(out MeasurementData displayValue, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
displayValue	O	Display value
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The displayed data were obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErNoData	Error	No value to obtain. *1
ErGetFailed	Error	Failed to obtain the display value.

Explanation:

This function obtains the data that are displayed on the instrument's display. The same color space data as the instrument's display value can be obtained.

* Blank data and the default value are stored in Id and Date, which are contained in the MeasurementData object that is obtained with this API.

*1. In this case, the calculated values are FLT_MAX.

4.6 Measurement Data (Stored in Device)

GetNumberOfSampleData()

Function:

Obtains the amount of measurement data stored in the instrument.

Format:

ReturnMessage **GetNumberOfSampleData**(out Int32 dataNum, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
dataNum	O	Number of stored measurement data
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The number of measurement data stored in the instrument was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErGetFailed	Error	Failed to obtain the number of measurement data stored in the instrument.

Explanation:

ReadSampleData()

Function:

Obtains the measurement data stored in the instrument. There are three overloaded functions (functions that have different arguments).

Format 1:

ReturnMessage **ReadSampleData**(Int32 dataNumber, [Desired color space class] sample, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
dataNumber	I	Sample data number
sample	I/O	Color space value data
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The measurement data were obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErNoData	Error	No specified data. *1
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErReadFailed	Error	Read failed.
ErCalcFailed	Error	Calculation failed. *2
ErOutOfRangeValue	Error	Calculation result is out of the chromaticity or luminance range. *3

Explanation:

Specify the variable of the color space class you want to use from the available color spaces as an argument.

*1. In this case, the calculated values are FLT_MAX.

*2. In this case, the calculated values are FLT_MIN.

*3. In this case, the calculated values are values out of the chromaticity or luminance range.

Format 2:

ReturnMessage **ReadSampleData**(Int32 dataNumber, [Desired color space class] sample, out Int32 targetCh, [Desired color space class] targetData, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
dataNumber	I	Sample data number
sample	I/O	Color space value data
targetCh	O	Target channel
target	I/O	Target data
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The measurement data was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErNoData	Error	No specified data. *1
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErReadFailed	Error	Read failed.
ErCalcFailed	Error	Calculation failed. *2
ErOutOfRangeValue	Error	Calculation result is out of the support range of chromaticity or luminance *3

Explanation:

Obtains the measurement data stored in the instrument, the data that was the target when that value was measured, and its target channel.

*1. In this case, the calculated values are FLT_MAX.

*2. In this case, the calculated values are FLT_MIN.

*3. In this case, the calculated values are the values out of the support range of chromaticity or luminance.

Format 3:

ReturnMessage **ReadSampleData**(Int32 dataNumber , [Desired color space class] sample, out Dictionary<string, object> detailedData, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
dataNumber	I	Sample data number
sample	I/O	Color space value data
detailedData	O	Target, desired calibration, measurement conditions data
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The measurement data was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErNoData	Error	No specified data. *1
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErReadFailed	Error	Read failed.
ErCalcFailed	Error	Calculation failed. *2
ErOutOfRangeValue	Error	Calculation result is out of the support range of chromaticity or luminance *3

Explanation:

Obtains the measurement data stored in the instrument, the target value, the calibration data, and the measurement conditions.

If there is no target value data and calibration data, that data is not stored in detailedData.

Example:

Measurement conditions

key:"PeakValley" value:"Peak"

Target data

key:"Target" value: MeasurementData type target data

Calibration data

key:"UserCalibrationData" value: ColorValueCalibData type user calibration value

*1. In this case, the calculated values are FLT_MAX.

*2. In this case, the calculated values are FLT_MIN.

*3. In this case, the calculated values are the values out of the support range of chromaticity or luminance.

DeleteSampleData()

Function:

Deletes the measurement data stored in the instrument.

Format:

ReturnMessage **DeleteSampleData**(Int32 dataNumber, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
dataNumber	I	Measurement data number to set -1: Delete all data
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The measurement data stored in the instrument was deleted.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErDeleteFailed	Error	Failed to delete the setting(s).

Explanation:

If -1 is specified for dataNumber, all measurement data stored in the instrument is deleted.

*1. If dataNumber is specified and deleted when there is no data, KmSuccess is returned.

4.7 Target Values

SetTargetCh()

Function:

Specifies the target channel setting for the instrument.

Format:

ReturnMessage **SetTargetCh**(Int32 targetCh, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
targetCh	I	Target channel
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The target channel that will be used was set.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	No data or invalid parameter in the specified target channel.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to set the target channel that will be used.

Explanation:

The target channel cannot be set if it has not been set with a target value.

GetTargetCh()

Function:

Obtains the target channel setting for the instrument.

Format:

ReturnMessage **GetTargetCh**(out Int32 targetCh, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
targetCh	O	Target channel
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The specified target channel was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErGetFailed	Error	Failed to obtain the specified target channel.

Explanation:

ReadTargetData()

Function:

Obtains the target value stored in the instrument.

Format:

ReturnMessage **ReadTargetData**(Int32 targetCh, "Desired color space class" target, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
targetCh	I	Target channel with the data to obtain
target	I/O	Color space value data
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The target value was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErNoData	Error	No data in the specified channel. *1
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErReadFailed	Error	Read failed.
ErCalcFailed	Error	Calculation failed. *2
ErOutOfRangeValue	Error	Calculation result is out of the support range of chromaticity or luminance *3

Explanation:

*1. In this case, the calculated values are FLT_MAX.

*2. In this case, the calculated values are FLT_MIN.

*3. In this case, the calculated values are the values out of the support range of chromaticity or luminance.

WriteTargetData()

Function:

Writes the target value to the instrument.

Format:

ReturnMessage **WriteTargetData**(Int32 targetCh, "Desired color space class" target, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
targetCh	I	Target channel
target	I	Target data
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The target value was set.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErWriteFailed	Error	Write failed.
ErOutOfRangeValue	Error	Calculation result is out of the support range of chromaticity or luminance *1

Explanation:

Enter the target value ID in target.Id.

You can use the below half-width characters within 12 as the target ID.

Alphanumeric [0～9] [a～z] [A～Z]

Symbol [!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~[space]]

*1. In this case, the calculated values are the values out of the support range of chromaticity or luminance.

DeleteTargetData()

Function:

Deletes the target value stored in the instrument.

Format:

ReturnMessage **DeleteTargetData**(Int32 targetCh, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
targetCh	I	Target channel to delete -1: Delete all data
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The target value was deleted.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErDeleteFailed	Error	Failed to delete the setting(s).

Explanation:

If -1 is specified for targetCh, all target values stored in the instrument will be deleted.

*1. If targetCh is specified and deleted when there is no data, KmSuccess is returned.

4.8 Measurement Time/Synchronization Settings

Class: MeasurementTime

Function:

Measurement time data class

Format:

```
class MeasurementTime
{
    double          ManualMeasurementTime;
    MeasTimeMode    MeasTimeMode;
}
```

Variable:

Variable	Explanation
ManualMeasurementTime	Manual measurement time
MeasTimeMode	Measurement time mode

Explanation:

Enum: MeasTimeMode

Value definitions for the measurement time mode

MeasTimeMode	**Definition value**	**Explanation**
Auto	0	Integration Time AUTO
Manual	1	Integration Time Manual

SetMeasurementTime()

Function:

Specifies the measurement time setting for the instrument.

Format:

ReturnMessage **SetMeasurementTime**(MeasurementTime measurementTime, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
measurementTime	I	Measurement time and measurement time mode
comport	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The measurement time was set.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to set the measurement time.

Explanation:

Set measurementTime.MeasTimeMode to MeasTimeMode.

To change only MeasTimeMode, set the measurement time with measurementTime.Time as the default value.

If MeasTimeMode is Manual (manual measurement), set measurementTime.Time to the

measurement time.
If MeasTimeMode is not Manual, measurementTime.Time is not reflected in the settings.

As you set the SyncMode to "Sync" by API function SetSyncMode, CS-150/CS-160 and LS-150/LS-160 execute measurement with the sync frequency regardless of MeasTimeMode setting.

GetMeasurementTime()

Function:

Obtains the measurement time setting for the instrument.

Return Type

ReturnMessage **GetMeasurementTime**(out MeasurementTime measurementTime, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
measurementTime	O	Measurement time and mode
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The measurement time was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErGetFailed	Error	Failed to obtain the measurement time.

Explanation:

Regardless of measurementTime.MeasTimeMode, the measurement time during a manual measurement is stored in measurementTime.Time.

Class: MeasurementFrequency

Function:

Synchronous/asynchronous measurement data class

Format:

```
class MeasurementFrequency
{
    double      Frequency;
    SyncMode    SyncMode;
}
```

Variable:

Variable	Explanation
frequency	Synchronization frequency when set to synchronous
syncMode	Synchronous/asynchronous

Explanation:

Enum: SyncMode

Value definitions for synchronous mode

SyncMode	**Definition value**	**Explanation**
Async	0	Synchronization Measurement OFF
Sync	1	Synchronization Measurement ON

SetSyncMode()

Function:

Specifies the synchronous/asynchronous measurement settings for the instrument.

Format:

ReturnMessage **SetSyncMode**(MeasurementFrequency measurementFrequency, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
measurementFrequency	I	Frequency and mode
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	Synchronous/asynchronous measurement was set.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to set the synchronous/asynchronous measurement settings.

Explanation:

Set measurementFrequency.SyncMode to SyncMode.

To change only SyncMode, set the synchronous/asynchronous measurement settings with measurementFrequency.Frequency as the default value.

If SyncMode is Sync (synchronous measurement), set the synchronization time in measurementFrequency.Frequency.

When measurementFrequency.SyncMode is Async, measurementFrequency.Frequency is not reflected in the settings.

GetSyncMode()

Function:

Obtains the synchronous/asynchronous measurement settings for the instrument.

Format:

ReturnMessage **GetSyncMode**(out MeasurementFrequency measurementFrequency, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
measurementFrequency	O	Synchronization frequency and synchronous measurement mode
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The synchronous/asynchronous measurement settings were obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErGetFailed	Error	Failed to set the synchronous/asynchronous measurement settings.

Explanation:

Regardless of measurementFrequency.SyncMode, the synchronization frequency during synchronous measurements is stored in measurementFrequency.Frequency.

4.9 Peak/Valley

Enum: PeakValley

Value definitions that represent the peaks/valley setting

PeakValley	Definition value	Explanation
OFF	0	OFF
Peak	1	Peak Measurement
Valley	2	Valley Measurement

SetPeakValley()

Function:

Specifies the peak/valley setting for the instrument.

Format:

ReturnMessage **SetPeakValley**(PeakValley peakValley, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
peakValley	I	Peak/Valley
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	Peak/valley was set.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to set peak/valley.

Explanation:

GetPeakValley()

Function:

Obtains the peak/valley setting for the instrument.

Format:

ReturnMessage **GetPeakValley**(out PeakValley peakValley, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
peakValley	O	Peak/Valley
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The peak/valley setting was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErGetFailed	Error	Failed to obtain the peak/valley setting.

Explanation:

4.10 LensSettings

Enum: CloseUpLensType

Value definitions for the close-up lens setting

CloseUpLensType	Definition value	Explanation
Standard	0	Standard (None)
No153	1	No.153
No135	2	No.135
No122	3	No.122
No110	4	No.110

SetCloseUpLens()

Function:

Specifies the close-up lens setting for the instrument.

Format:

ReturnMessage **SetCloseUpLens**(CloseUpLensType closeUpLensType, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
closeUpLensType	I	Close-up lens type
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The close-up lens was set.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to set the close-up lens.

Explanation:

GetCloseUpLens()

Function:

Obtains the close-up lens setting for the instrument.

Format:

ReturnMessage **GetCloseUpLens**(out CloseUpLensType closeUpLensType, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
closeUpLensType	O	Close-up lens setting
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The close-up lens setting was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErGetFailed	Error	Failed to set the close-up lens.

Explanation:

4.11 User Calibration Channel

SetCalibrationCh()

Function:

Specifies the user calibration channel for the instrument.

Format:

ReturnMessage **SetCalibrationCh**(Int32 calibrationCh, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
calibrationCh	I	User calibration channel
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The user calibration channel that will be used was set.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	No data or invalid parameter in the specified user calibration channel.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to set the user calibration channel that will be used.

Explanation:

The user calibration channel cannot be set if it has not been set with a user calibration value.

GetCalibrationCh()

Function:

Obtains the user calibration channel set on the instrument.

Format:

ReturnMessage **GetCalibrationCh**(out Int32 calibrationCh, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
calibrationCh	O	User calibration channel
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The specified user calibration channel was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErGetFailed	Error	Failed to obtain the specified user calibration channel.

Explanation:

4.12 User Calibration

Class: UserCalibData

Function:

User calibration data class

Format:

```
class UserCalibData
{
    string              Id;
    DateTime            Date;
    CalibType           CalibType;
    List<double>        Coef;
}
```

Variable:

Variable	Explanation
Id	ID
Date	Registration date/time
CalibType	Calibration type
Coef	Calibration coefficients

Explanation:

Enum: CalibType

Value definitions that represent the calibration type

CalibType	**Definition value**	**Explanation**
OnePoint	0	Single-point calibration
RGB	1	RGB calibration
WRGB	2	WRGB calibration

SetMatrixCalib()

Function:

Sets the user calibration coefficients on the instrument.

Format:

ReturnMessage **SetMatrixCalib**(Int32 calibrationCh, List<"Desired color space class"> measurementDataList, List<"Desired color space class"> correctDataList, CalibType type,string id, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
calibrationCh	I	User calibration channel
measurementDataList	I	Measurement value (before calibration value) list
correctDataList	I	True value (after calibration value) list
type	I	Calibration type (Single-point calibration, RGB calibration, WRGB calibration)
id	I	ID
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The user calibration coefficients were set.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to set the user calibration coefficients.

ErOutOfRangeValue	Error	Calculation result is out of the support range of chromaticity or luminance *1

Explanation:

Single-point calibration

type = CalibType.One Point

measurementDataList[0] = color space value that measured W

Set correctDataList to the same color space value as measurementDataList.

*If you use a small value near to zero for the single point calibration, the calibration coefficient are set to 1 since the calculation is impossible.

RGB calibration

type = CalibType.RGB

measurementDataList[0] = color space value that measured R

measurementDataList[1] = color space value that measured G

measurementDataList[2] = color space value that measured B

Set correctDataList to the same color space value as measurementDataList.

WRGB calibration

type = CalibType.WRGB

measurementDataList[0] = color space value that measured W

measurementDataList[1] = color space value that measured R

measurementDataList[2] = color space value that measured G

measurementDataList[3] = color space value that measured B

Set correctDataList to the same color space value as measurementDataList.

*In the RGB calibration or the WRGB calibration, the inverse matrix does not occasionally exist. In this case, ErSetFailed would return as errorCode and the message "Inverse matrix does not exist." is in errorMessage.

The calibration date/time is automatically written with the PC's time.

You can use the below half-width characters within 12 as the ID.

Alphanumeric [0～9] [a～z] [A～Z]

Symbol [!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~[space]]

*1. In this case, the calculated values are the values out of the support range of chromaticity or luminance.

GetCalibData()

Function:

Obtains the calibration parameters set for the user calibration channel on the instrument.

Format:

ReturnMessage **GetCalibData**(Int32 calibrationCh, out List<MeasurementData> measurementDataList, out List<MeasurementData> correctDataList, out UserCalibData calibData, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
calibrationCh	I	User calibration channel
measurementDataList	O	Measurement values used when setting the user calibration coefficients
correctDataList	O	True values used when setting the user calibration coefficients
calibData	O	User calibration data that will be obtained
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The user calibration coefficients were obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErNoData	Error	No data in the specified channel. *1
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErGetFailed	Error	Failed to obtain the user calibration coefficients.

ErCalcFailed	Error	Calculation failed. *2
ErOutOfRangeValue	Error	Calculation result is out of the support range of chromaticity or luminance *3

Explanation:

・ The luminance unit for the obtained measurement values and true values is the same as the setting on the instrument.

*1. In this case, the calculated values are FLT_MAX.

*2. In this case, the calculated values are FLT_MIN.

*3. In this case, the calculated values are the values out of the support range of chromaticity or luminance.

DeleteCalibData()

Function:

Deletes the user calibration coefficients stored in the instrument.

Format:

ReturnMessage **DeleteCalibData**(Int32 calibrationCh, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
calibrationCh	I	Calibration channel to delete -1: Delete all data
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The user calibration coefficients were deleted.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErDeleteFailed	Error	Failed to delete the user calibration coefficients.

Explanation:

If -1 is specified for calibrationCh, all of the user calibration coefficients stored in the instrument will be deleted.

*1. If calibrationCh is specified and deleted when there is no data, KmSuccess is returned.

4.13 CCF (Color Correction Factor)

If the color correction factor for the light source to measure is known ahead of time, it can be set on the instrument to allow corrected measurements to be executed.

Class: ColorCorrectionFactor

Function:

Color correction factor data class

Format:

```
class ColorCorrectionFactor
{
    double      Coef;
    CCFMode     CcfMode;
}
```

Variable:

Variable	Explanation
Coef	Color correction factor value
CcfMode	ON/OFF

Explanation:

Enum: CCFMode

CCFMode	**Definition value**	**Explanation**
OFF	0	OFF
ON	1	ON

SetCCF()

Function:

Specifies the CCF (color correction factor) setting.

Format:

ReturnMessage **SetCCF**(ColorCorrectionFactor ccfData, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
ccfData	I	Color correction factor value and on/off
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The color correction factor was set and turned on or off.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to set the color correction factor and on/off settings.

Explanation:

To only turn the color correction factor on or off, set the color correction factor settings with ccfData.Ccf as the default value.

GetCCF()

Function:

Obtains the CCF (color correction factor) setting.

Format:

ReturnMessage **GetCCF**(out ColorCorrectionFactor ccfData, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
ccfData	O	Color correction factor value and on/off
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The color correction factor and on/off settings were obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErGetFailed	Error	Failed to obtain the color correction factor and on/off settings.

Explanation:

4.14 Power Settings

Enum: AutoPowerOff

Value definitions for the auto power off setting

AutoPowerOff	Definition value	Explanation
Off	0	Auto power off: Off
On	1	Auto power off: On

SetAutoPowerOff()

Function:

Specifies the auto power off setting for the instrument.

Format:

ReturnMessage **SetAutoPowerOff**(AutoPowerOff autoPowerOff, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
autoPowerOff	I	Auto power off setting
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	Auto power off was set.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to set auto power off.

Explanation:

GetAutoPowerOff()

Function:

Obtains the auto power off setting for the instrument.

Format:

ReturnMessage **GetAutoPowerOff**(out AutoPowerOff autoPowerOff, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
autoPowerOff	O	Auto power off setting
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	Auto power off was set.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErGetFailed	Error	Failed to obtain the auto power off setting.

Explanation:

4.15 Backlight

Enum: BackLightMode

Value definitions for the backlight setting

BackLightMode	Definition value	Explanation
Off	0	Backlight Off
On	1	Backlight On

SetBackLightOnOff()

Function:

Specifies the on/off setting for the instrument's backlight.

Format:

ReturnMessage **SetBackLightOnOff**(BackLightMode backLightMode, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
backLightMode	I	Backlight On/Off
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The backlight was turned on or off.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErSetFailed	Error	Failed to set the setting.

Explanation:

GetBackLightOnOff()

Function:

Specifies the on/off setting for the instrument's backlight.

Format:

ReturnMessage **GetBackLightOnOff**(out BackLightMode backLightMode, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
backLightMode	O	Backlight On/Off
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The backlight on/off setting was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErGetFailed	Error	Failed to obtain the backlight on/off setting.

Explanation:

Enum: BackLightLevel

Value definitions for the backlight setting

BackLightLevel	**Definition value**	**Explanation**	
Level1	1	Brightness 1	Dark ↕ Bright
Level2	2	Brightness 2	
Level3	3	Brightness 3	
Level4	4	Brightness 4	
Level5	5	Brightness 5	

SetBackLightLevel()

Function:

Specifies the brightness setting for the instrument's backlight.

Format:

ReturnMessage **SetBackLightLevel**(BackLightLevel backLightLevel, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
backLightLevel	I	Backlight brightness
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The backlight brightness was set.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to set the backlight brightness.

Explanation:

GetBackLightLevel()

Function:

Obtains the brightness setting for the instrument's backlight.

Format:

ReturnMessage **GetBackLightLevel**(out BackLightLevel backLightLevel, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
backLightLevel	O	Backlight setting
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The backlight brightness setting was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErGetFailed	Error	Failed to obtain the backlight brightness setting.

Explanation:

4.16 Display Digits

SetColorDispDigit()

Function:

Specifies the color display digits setting for the instrument.

Format:

ReturnMessage **SetColorDispDigit**(Int16 digitNum, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
digitNum	I	Color display digits
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The color display digits were set.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to set the color display digits.

Explanation:

To set 3 digits, store digitNum = 3;

To set 4 digits, store digitNum = 4;

GetColorDispDigit()

Function:

Obtains the color display digits setting for the instrument.

Format:

ReturnMessage **GetColorDispDigit**(out Int16 digitNum, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
digitNum	O	Color display digits
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The color display digits setting was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErSetFailed	Error	Failed to set the color display digits.

Explanation:

4.17 Display Type (Absolute Value/Difference/Ratio)

Enum: DispType

Value definitions that represent the display type setting on the instrument's screen

DispType	Definition value	Explanation
Abs	0	Absolute value display
Diff	1	Differential display
Ratio	2	Ratio display

SetDisplayType()

Function:

Specifies the display type setting (absolute value, differential, ratio) on the instrument's display.

Format:

ReturnMessage **SetDisplayType**(DispType dispType, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
dispType	I	Display type
comport	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The display type was set.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to set the setting.

Explanation:

GetDisplayType()

Function:

Obtains the display type setting (absolute value, differential, ratio) on the instrument's display.

Format:

ReturnMessage **GetDisplayType**(out DispType dispType, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
dispType	O	Display type
comport	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The display type was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErGetFailed	Error	Failed to obtain the display type.

Explanation:

4.18 Language/Date/Time

Enum: DisplayLanguage

These are the value definitions that are used when setting and obtaining the language using Set/GetLanguage().

DisplayLanguage	Definition value	Explanation
ENG	0	English (US)
JPN	1	Japanese
CHI	2	Chinese (simplified)

SetDisplayLanguage()

Function:

Specifies the language setting for the instrument.

Format:

ReturnMessage **SetDisplayLanguage**(DisplayLanguage displayLanguage, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
displayLanguage	I	Language
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The specified language was set.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to set the language.

Explanation:

GetDisplayLanguage()

Function:

Obtains the language setting for the instrument.

Format:

ReturnMessage **GetDisplayLanguage**(out DisplayLanguage displayLanguage, Int32 comPort)

Argument:

Argument	I/O	Explanation
displayLanguage	O	Language
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The language setting was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErGetFailed	Error	Failed to obtain the language setting.

Explanation:

Class: DateTime

Function:

Date/time class

This class is provided by the System namespace.

Format:

https://msdn.microsoft.com/en-US/library/system.datetime(v=vs.110).aspx

Explanation:

Use when setting and obtaining the date and time.

SetDateTime()

Function:

Specifies the date/time settings for the instrument.

Format:

ReturnMessage **SetDateTime**(DateTime dateTime, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
dateTime	I	Date/time
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The date/time was set.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to set the date/time.

Explanation:

Please set the year, month, day, hour, minute, and second in dateTime.

GetDateTime()

Function:

Obtains the date/time setting for the instrument.

Format:

ReturnMessage **GetDateTime**(out DateTime dataTime, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
dataTime	O	Date/time
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	Success
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErGetFailed	Error	Failed to obtain the setting.

Explanation:

Enum: DateFormat

Value definitions for the date format

DateFormat	Definition value	Explanation
YYMMDD	0	Year/month/day
MMDDYY	1	Month/day/year
DDMMYY	2	Day/month/year

SetDateFormat()

Function:

Specifies the date format settings for the instrument.

Format:

ReturnMessage **SetDateFormat**(DateFormat dateFormat, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
dateFormat	I	Date format
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The date format was set.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to set the date format.

Explanation:

GetDateFormat()

Function:

Obtains the date format settings for the instrument.

Format:

ReturnMessage **GetDateFormat**(out DateFormat dateFormat, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
dateFormat	O	Date format
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The date format was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErGetFailed	Error	Failed to obtain the date format.

Explanation:

4.19 Device Color Mode

Enum: ColorMode

Value definitions that represent the color mode displayed on the instrument

ColorMode	Definition value	Explanation
Lvxy	0	Lv, x, y
Lvudvd	1	Lv, u', v'
LvTcpDuv	2	Lv, Tcp, duv
XYZ	3	X, Y, Z
LvDwPe	4	Lv, dominant wavelength λd, excitation purity Pe
Lv	5	Lv

SetColorMode()

Function:

Specifies the color mode setting.

Format:

ReturnMessage SetColorMode(ColorMode colorMode, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
colorMode	I	Color mode
comport	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The color mode was set.
ErNoConnect	Error	No instrument connected to the specified virtual COM port.
ErInvalidParameter	Error	The ColorMode in the parameter colorMode is an invalid parameter or the setting of it in the instrument is off.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to set the setting.

Explanation:

Sets the color mode to display on the instrument's display.

You need to turn on the color mode display setting in the parameter colorMode by calling SetColorModeDisplayOnOff() before you set it.

GetColorMode()

Function:

Obtains the color mode setting.

Format:

ReturnMessage GetColorMode(out ColorMode colorMode, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
colorMode	O	Color mode
comport	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The color mode was obtained.
ErNoConnect	Error	No instrument connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErCannotCommand	Error	This model does not support the specified command.
ErGetFailed	Error	Failed to set the setting.

Explanation:

Obtains the color mode to display on the instrument's display.

Enum: ColorModeDisplay

ColorModeDisplay	Definition value	Explanation
Off	0	OFF
On	1	ON

SetColorModeDisplayOnOff()

Function:

Turns on/off an desired color mode to display on the instrument's display.

Format:

ReturnMessage **SetColorModeDisplayOnOff**(ColorMode colorMode, ColorModeDisplay onOff, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
colorMode	I	Color mode
onOff	I	OnOff
comport	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The desired color mode was turned on or off.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to turn the desired color mode on or off.

Explanation:

For the colorMode argument, you can only set a color mode that the connected model can handle.

GetColorModeDisplayOnOff()

Function:

Obtains the on/off setting for an desired color mode to display on the instrument's display.

Format:

ReturnMessage **GetColorModeDisplayOnOff**(ColorMode colorMode, out ColorModeDisplay onOff, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
colorMode	I	Color mode
onOff	O	OnOff
comport	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The desired color mode on/off setting was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErCannotCommand	Error	This model does not support the specified command.
ErGetFailed	Error	Failed to obtain the desired color mode on/off setting.

Explanation:

4.20 Data Save Mode

Enum: DataSaveMode

DataSaveMode	Definition value	Explanation
AutoSave	0	Auto
ManuSave	1	Manual

SetDataSaveMode()

Function:

Specifies the data save mode setting for the instrument.

Format:

ReturnMessage **SetDataSaveMode**(DataSaveMode dataSaveMode, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
dataSaveMode	I	Data save mode
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The data save mode was set.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to set the data save mode.

Explanation:

GetDataSaveMode()

Function:

Obtains the data save mode setting for the instrument.

Format:

ReturnMessage **GetDataSaveMode**(out DataSaveMode dataSaveMode, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
dataSaveMode	O	Data save mode
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The data save mode was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErGetFailed	Error	Failed to obtain the data save mode setting.

Explanation:

4.21 Periodic Calibration Settings

Enum: PeriodicCalibNotify

These are the value definitions that are used when setting and obtaining the periodic calibration alert settings using Set/GetPeriodicCalibNotify().

PeriodicCalibNotify	Definition value	Explanation
On	0	Calibration Alert ON
Off	1	Calibration Alert OFF

SetPeriodicCalibNotify()

Function:

Turns the periodic calibration alert for the instrument on or off.

Format:

ReturnMessage **SetPeriodicCalibNotify**(PeriodicCalibNotify periodicCalibNotify, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
periodicCalibNotify	I	Periodic calibration alert setting
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The periodic calibration alert was turned on or off.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to turn the periodic calibration

		alert on or off.

Explanation:

GetPeriodicCalibNotify()

Function:

Obtains the periodic calibration alert on/off setting.

Format:

ReturnMessage GetPeriodicCalibNotify(out PeriodicCalibNotify periodicCalibNotify, Int32 comPort)

Argument:

Argument	I/O	Explanation
periodicCalibNotify	O	Periodic calibration alert setting
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The periodic calibration alert on/off setting was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErGetFailed	Error	Failed to obtain the periodic calibration alert on/off setting.

Explanation:

4.22 Button Settings

Enum: ToggleStatus

Value definitions for the toggle setting

ToggleStatus	Definition value	Explanation
Off	0	Measurement button toggle off (Standard)
On	1	Measurement button toggle on (Toggle)

SetToggleOnOff()

Function:

Specifies the toggle on/off setting for the instrument.

Format:

ReturnMessage **SetToggleOnOff**(ToggleStatus toggleStatus, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
toggleStatus	I	Toggle setting
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The measurement button toggle was turned on or off.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to turn the measurement button toggle on or off.

Explanation:

When ToggleStatus is on,

press the measurement button to start the measurement and then press the measurement button one more time to stop the measurement.

When ToggleStatus is off,

press the measurement button to start the measurement and keep holding the measurement button down to continue the measurement, then release the measurement button to stop the measurement.

GetToggleOnOff()

Function:

Obtains the toggle on/off setting for the instrument.

Format:

ReturnMessage **GetToggleOnOff**(out ToggleStatus toggleStatus, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
toggleStatus	O	Toggle setting
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The measurement button toggle on/off setting was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErGetFailed	Error	Failed to obtain the measurement button toggle on/off setting.

Explanation:

Enum: TriggerStatus

Value definitions for the button status

TriggerStatus	Definition value	Explanation
Off	0	Measurement button operation disabled
On	1	Measurement button operation enabled

SetTriggerOnOff()

Function:

Specifies the enabled/disabled setting for the measurement button on the instrument.

Format:

ReturnMessage **SetTriggerOnOff**(TriggerStatus triggerStatus, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
triggerStatus	I	Measurement button enabled/disabled
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The measurement button enabled/disabled setting was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErSetFailed	Error	Failed to obtain the measurement button enabled/disabled setting.

Explanation:

If TriggerStatus is off, measurements cannot be taken using the instrument's trigger

(measurement button).
The instrument's trigger is to be disabled automatically after you disconnect the instrument.
You need to call this API on every connection if you want to measure by using the instrument's trigger.

GetTriggerOnOff()

Function:

Specifies the enabled/disabled setting for the measurement button on the instrument.

Format:

ReturnMessage **GetTriggerOnOff**(out TriggerStatus triggerStatus, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
triggerStatus	O	Measurement button enabled/disabled
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The measurement button enabled/disabled setting was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErGetFailed	Error	Failed to obtain the setting.

Explanation:

This function obtains the measurement button enabled/disabled setting.

4.23 Device/SDK Information

Class: DeviceInfo

Function:

Instrument information class

Format:

```
class DeviceInfo
{
    string              ProductName;
    string              SerialNumber;
    string              SoftMajorVersion;
    string              SoftMinorVersion;
    string              SoftFreeVersion;
    DateTime            PeriodicCalibrationExpirationDate;
    bool                PeriodicCalibrationWarningStatus;
}
```

Variable:

Variable	Explanation
ProductName	Product name.
SerialNumber	Stores the instrument serial number.
SoftMajorVersion	Stores the firmware major version information.
SoftMinorVersion	Stores the firmware minor version information.
SoftFreeVersion	Stores the firmware free version information.
PeriodicCalibrationExpirationDate	Stores the periodic calibration expiration date.
PeriodicCalibrationWarningStatus	Stores the bool status of the periodic calibration warning. If this API returns true, you should send the instrument to service for periodic calibration.

Explanation:

This class is used when obtaining the instrument information using GetDeviceInfo().

GetDeviceInfo()

Function:

Obtains instrument information including the product name and the periodic calibration expiration date.

Format:

ReturnMessage **GetDeviceInfo**(out DeviceInfo deviceInfo, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
deviceInfo	O	Instrument information
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The instrument information was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in the middle of a process.
ErGetFailed	Error	Failed to obtain the instrument information.

Explanation:

GetSDKVersion()

Function:

Obtains the SDK version information.

Format:

ReturnMessage **GetSDKVersion**(out string SDKFileVersion)

Argument:

Argument	I/O	Explanation
SDKFile	O	O: SDK version

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The version information was obtained.
ErGetFailed	Error	Failed to obtain the setting.

Explanation:

The SDK version is output as a string like "LC-MISDK : 1.0.0.0".

4.24 Measurement Unit Settings

Enum: LuminanceUnit

Defined values that represent the luminance unit

LuminanceUnit	Definition value	Explanation
cdm2	0	cd/m^2
other	1	fL

SetLuminanceUnit()

Function:

Specifies the luminance unit setting for the instrument.

Format:

ReturnMessage **SetLuminanceUnit**(LuminanceUnit luminanceUnit, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
luminanceUnit	I	Luminance unit
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The luminance unit was set.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	Error	Invalid parameter.
ErCannotCommand	Error	This model does not support the specified command.
ErInstrumentProcessing	Error	The instrument cannot receive any commands because it is in processingin the middle of a process.
ErSetFailed	Error	Failed to set the luminance unit.

Explanation:

* The luminance unit cannot be specified when the instrument language setting is Japanese.
In this case, ErInvalidParameter is returned.

GetLuminanceUnit()

Function:

Specifies the luminance unit setting for the instrument.

Format:

ReturnMessage **GetLuminanceUnit**(out LuminanceUnit luminanceUnit, Int32 comPort = 0)

Argument:

Argument	I/O	Explanation
luminanceUnit	O	Luminance unit
comPort	I	Virtual COM Port number

Return Value:

Return value	Type	Explanation
KmSuccess	Success	The luminance unit was obtained.
ErNoConnect	Error	No instrument is connected to the specified virtual COM port.
ErCannotCommand	Error	This model does not support the specified command.
ErGetFailed	Error	Failed to obtain the luminance unit.

Explanation:

5. Appendix

5.1 Error Code List

ErrorDefine	Definition value	Cause
KmSuccess	0	The processing was completed normally.
ErNoConnect	10	No instrument is connected to the specified virtual COM port.
ErInvalidParameter	25	The assigned parameter is incorrect.
ErCannotCommand	30	This model does not support the specified command.
ErNoData	45	No specified data.
ErOutOfRangeValue	50	Calculation result is out of the support range of chromaticity or luminance.
ErInstrumentProcessing	60	The instrument cannot receive any commands because it is in the middle of a process.
ErConnectFailed	100	Failed to connect to the instrument.
ErDisConnectFailed	110	There are one or more ports that cannot be disconnected.
ErSetFailed	120	Failed to set the setting.
ErGetFailed	130	Failed to obtain the setting.
ErCalcFailed	140	Calculation failed.
ErCancelFailed	150	Cancellation failed.
ErWriteFailed	160	Write failed.
ErReadFailed	170	Read failed.
ErDeleteFailed	180	Failed to delete the setting(s).
ErMeasurementFailed	200	Measurement failed.

5.2 Color Space Class

The color spaces that can be handled by the SDK differ by the model of instrument.

Class: MeasurementData

Function:

Color space base class

Format:

```
class MeasurementData
{
    string    Id;
    ColorMode ColorModeId;
    DateTime    Date;
    Dictionary<string, double> ColorSpaceValue
}
```

Variable:

Variable	Explanation
Id	ID
ColorModeId	Color space ID
Date	Registration/measurement date/time
ColorSpaceValue	Color space value

Explanation:

Class: XYZ

Function:

$(\mathrm{X}\ \ Y\ \ Z)$ Class

Format:

```
class XYZ : MeasurementData
{
public XYZ(double X = 0.0, double Y = 0.0, double Z = 0.0)
        {
            ColorModeId = ColorMode.XYZ;
            ColorSpaceValue.Add("X", X);
            ColorSpaceValue.Add("Y", Y);
            ColorSpaceValue.Add("Z", Z);
        }

        public double X
        {
            get { return ColorSpaceValue["X"]; }
            set { ColorSpaceValue["X"] = value; }
        }
        public double Y
        {
            get { return ColorSpaceValue["Y"]; }
            set { ColorSpaceValue["Y"] = value; }
        }
        public double Z
        {
            get { return ColorSpaceValue["Z"]; }
            set { ColorSpaceValue["Z"] = value; }
        }
}
```

Variable:

Variable	Explanation
X	tristimulus value X
Y	tristimulus value Y
Z	tristimulus value Z

Explanation:

Set/obtain color space values one at a time

```
XYZ xyz = new XYZ();
xyz.X = 10; xyz.Y = 20; xyz.Z = 30;
```

Obtain color space values with foreach

```
foreach(var colorValue in xyz.ColorSpaceValue){
    WriteLine(colorValue.Value);
}
```

Class: Lvxy

Function:

$(L_v \ x \ y)$ Class

Format:

```
class Lvxy : MeasurementData
{
public Lvxy(LuminanceUnit unit = LuminanceUnit.cdm2, double Lv = 0.0, double x =
0.0, double y = 0.0)
        {
            this.unit = unit;
            this.ColorModeId = ColorMode.Lvxy;
            this.ColorSpaceValue.Add("Lv", Lv);
            this.ColorSpaceValue.Add("x", x);
            this.ColorSpaceValue.Add("y", y);
        }
        public double Lv
        {
            get { return ColorSpaceValue["Lv"]; }
            set { ColorSpaceValue["Lv"] = value; }
        }
        public double x
        {
            get { return ColorSpaceValue["x"]; }
            set { ColorSpaceValue["x"] = value; }
        }
        public double y
        {
            get { return ColorSpaceValue["y"]; }
            set { ColorSpaceValue["y"] = value; }
        }

        public LuminanceUnit unit;

}
```

Variable:

Variable	Explanation
Lv	Luminance
x	chromacity x
y	chromacity y
unit	Luminance unit

Explanation:

Set the luminance unit : cd/m^2

Lvxy lvxy = new Lvxy();

or Lvxy lvxy = new Lvxy(LuminanceUnit.cdm2);

Set the luminance unit : fL

Lvxy lvxy = new lvxy(LuminanceUnit.other);

Class: Lvudvd

Function:

$(L_v \ \ u' \ \ v')$ Class

Format:

```
public class Lvudvd : MeasurementData
    {

        public Lvudvd(LuminanceUnit unit = LuminanceUnit.cdm2, double Lv = 0.0,
double ud = 0.0, double vd = 0.0)
        {
            this.unit = unit;
            ColorModeId = ColorMode.Lvudvd;
            ColorSpaceValue.Add("Lv", Lv);
            ColorSpaceValue.Add("ud", ud);
            ColorSpaceValue.Add("vd", vd);
        }

        public double Lv
        {
            get { return ColorSpaceValue["Lv"]; }
            set { ColorSpaceValue["Lv"] = value; }
        }
        public double ud
        {
            get { return ColorSpaceValue["ud"]; }
            set { ColorSpaceValue["ud"] = value; }
        }
        public double vd
        {
            get { return ColorSpaceValue["vd"]; }
            set { ColorSpaceValue["vd"] = value; }
        }

        public LuminanceUnit unit;
    }
```

Variable:

Variable	Explanation
Lv	Luminance
ud	CIE 1976 UCS chromaticity coordinates u'
vd	CIE 1976 UCS chromaticity coordinates v'
unit	Luminance unit

Explanation:

Set the luminance unit： cd/m^2

Lvudvd lvudvd = new Lvudvd();

or Lvudvd lvudvd = new Lvudvd(LuminanceUnit.cdm2);

Set the luminance unit： fL

Lvudvd lvudvd = new Lvudvd(LuminanceUnit.other);

Class: LvTcpDuv

Function:

(Luminance, correlated color temperature, color difference from black body trajectory) class

Format:

```
public class LvTcpDuv : MeasurementData
{
    public LvTcpDuv(LuminanceUnit unit = LuminanceUnit.cdm2,double Lv = 0.0,
double Tcp = 0.0, double Duv = 0.0)
    {
        this.unit = unit;
        ColorModeId = ColorMode.LvTcpDuv;
        ColorSpaceValue.Add("Lv", Lv);
        ColorSpaceValue.Add("Tcp", Tcp);
        ColorSpaceValue.Add("Duv", Duv);
    }

    public double Lv
    {
        get { return ColorSpaceValue["Lv"]; }
        set { ColorSpaceValue["Lv"] = value; }
    }
    public double Tcp
    {
        get { return ColorSpaceValue["Tcp"]; }
        set { ColorSpaceValue["Tcp"] = value; }
    }
    public double duv
    {
        get { return ColorSpaceValue["Duv"]; }
        set { ColorSpaceValue["Duv"] = value; }
    }

    public LuminanceUnit unit;
}
```

Variable:

Variable	Explanation
Lv	Luminance
Tcp	Correlated color temperature T
Duv	Color difference from the black body locus duv
unit	Luminance unit

Explanation:

Set the luminance unit : cd/m^2

LvTcpDuv lvTcpDuv = new LvTcpDuv();

or LvTcpDuv lvTcpDuv = new LvTcpDuv(LuminanceUnit.cdm2);

Set the luminance unit : fL

LvTcpDuv lvTcpDuv = new LvTcpDuv(LuminanceUnit.other);

*1. When color temperature data is written to the instrument using the APIs such as WriteTargetData and SetMatrixCalib, a calculation error will occur on certain models (CS-150 series, others). This occurs because a conversion calculation error when writing the data cannot be avoided because a portion of the models manage the data in XYZ format.
It is recommended to use duv in the range from -0.02 to +0.02 to suppress the error.

Class: LvDwPe

Function:

(Luminance, dominant wavelength, excitation purity) class

Format:

```
public class LvDwPe : MeasurementData
{
    public LvDwPe(LuminanceUnit unit = LuminanceUnit.cdm2,double Lv = 0.0,
double Dw = 0.0, double Pe = 0.0)
    {
        this.unit = unit;
        ColorModeId = ColorMode.LvDwPe;
        ColorSpaceValue.Add("Lv", Lv);
        ColorSpaceValue.Add("Dw", Dw);
        ColorSpaceValue.Add("Pe", Pe);
    }

    public double Lv
    {
        get { return ColorSpaceValue["Lv"]; }
        set { ColorSpaceValue["Lv"] = value; }
    }
    public double Dw
    {
        get { return ColorSpaceValue["Dw"]; }
        set { ColorSpaceValue["Dw"] = value; }
    }
    public double Pe
    {
        get { return ColorSpaceValue["Pe"]; }
        set { ColorSpaceValue["Pe"] = value; }
    }

    public LuminanceUnit unit;
}
```

Variable:

Variable	Explanation
Lv	Luminance
Dw	Dominant wavelength [nm]
Pe	Excitation purity [%]
unit	Luminance unit

Explanation:

Set the luminance unit : cd/m^2

LvDwPe lvDwPe = new LvDwPe();

or LvDwPe lvDwPe = new LvDwPe(LuminanceUnit.cdm2);

Set the luminance unit： fL

```
LvDwPe lvDwPe = new LvDwPe(LuminanceUnit.other);
```

*1 If the dominant wavelength (Dw) is negative, this indicates a complementary dominant wavelength.

Class: Lv

Function:

Luminance class

Format:

```
public class Lv : MeasurementData
{
    public Lv(LuminanceUnit unit = LuminanceUnit.cdm2,double lv = 0.0)
    {
        this.unit = unit;
        ColorModeId = ColorMode.Lv;
        ColorSpaceValue.Add("Lv", lv);
    }
    public double lv
    {
        get { return ColorSpaceValue["Lv"]; }
        set { ColorSpaceValue["Lv"] = value; }
    }

    public LuminanceUnit unit;
}
```

Variable:

Variable	Explanation
Lv	Luminance
unit	Luminance unit

Explanation:

Set the luminance unit : cd/m^2

Lv lv = new Lv(); or Lv lv = new Lv(LuminanceUnit.cdm2);

Set the luminance unit : fL

Lv lv = new lv(LuminanceUnit.other);

5.3 Device driver install

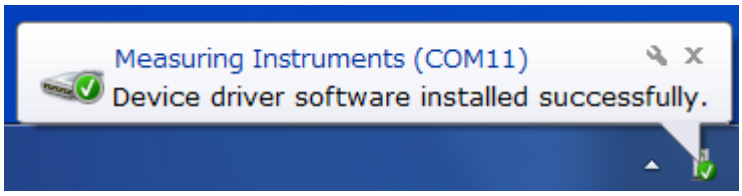
You have to install the device driver to connect to the instrument via USB.

At first, connect USB cable between PC and the instrument, and power on the instrument. Then, the driver installation automatically starts.

The device driver automatically installed by Windows10 may not work well. Therefore, please install the device driver file "KMMIUSB.INF" manually by following below installation guide.

Automatic installation

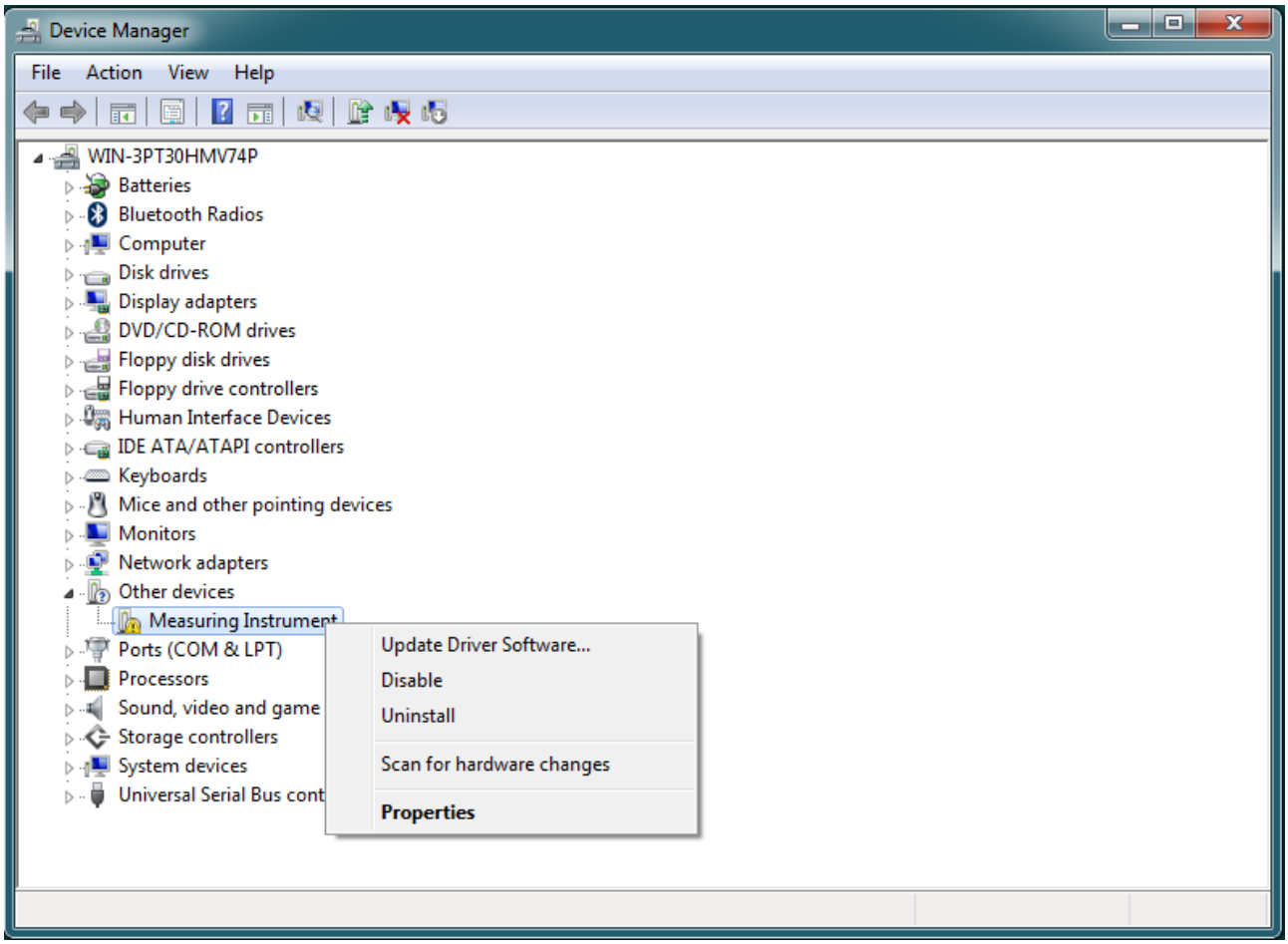
If the automatic installation is succeeded as below, the installation process is completed here.



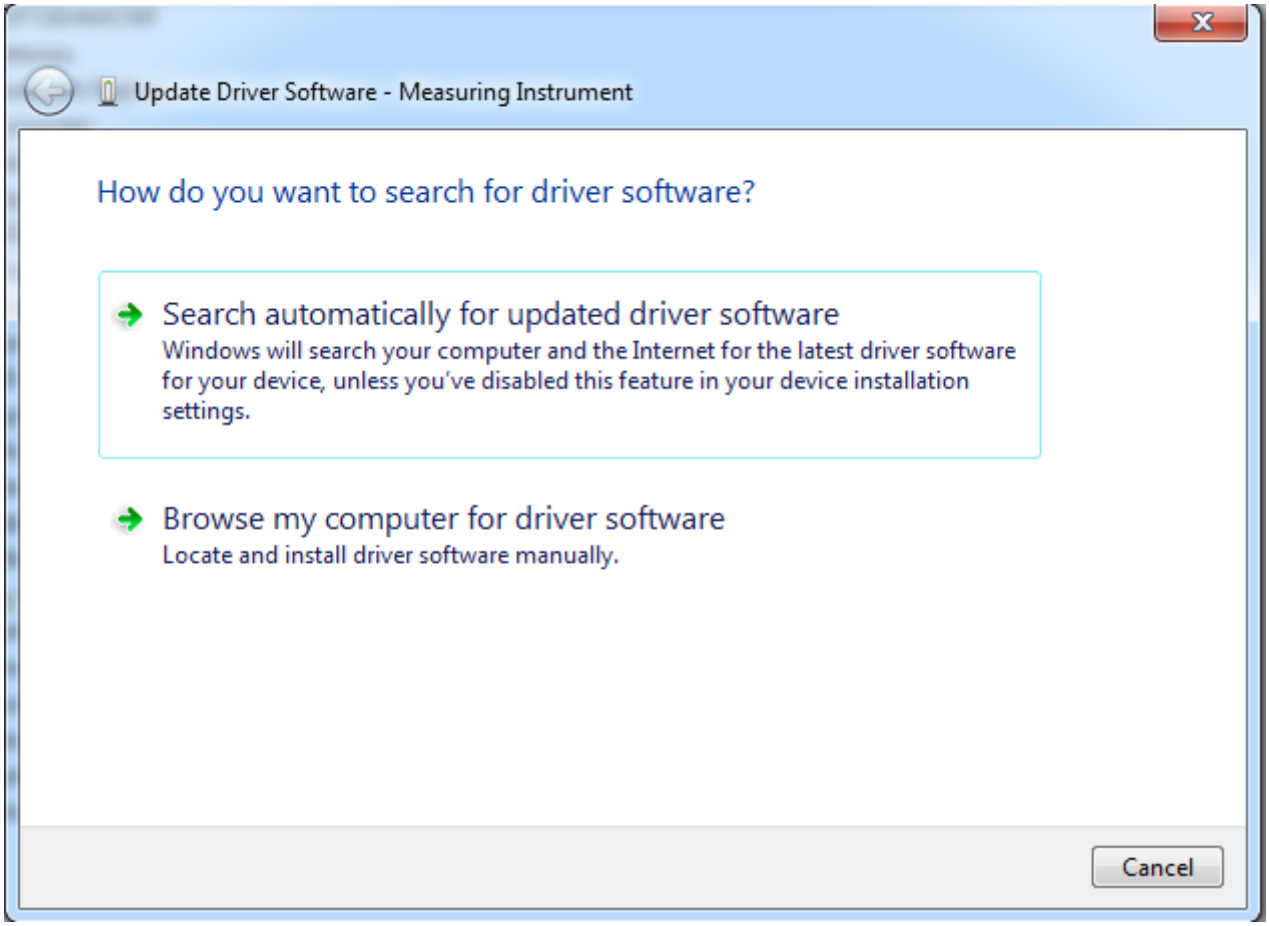
Manual installation

When the automatic installation is failed as below, please execute manual installation along the following procedure.

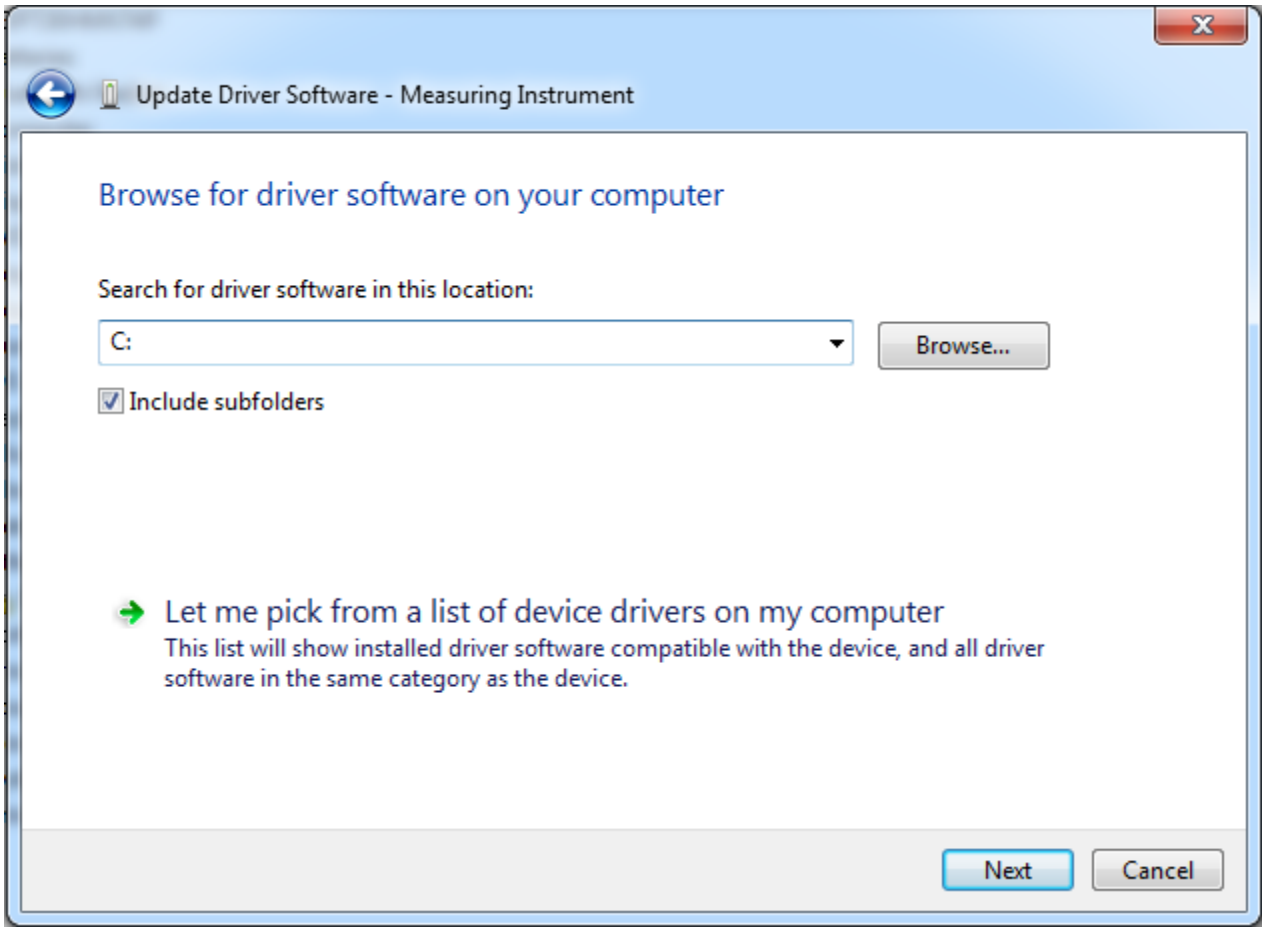
Open device manager, right click [Measuring Instruments] in [Other devices] and click "Update Driver Software." In the case that the driver installation is not succeeded, caution icon is attached to [Measuring Instruments] as below.



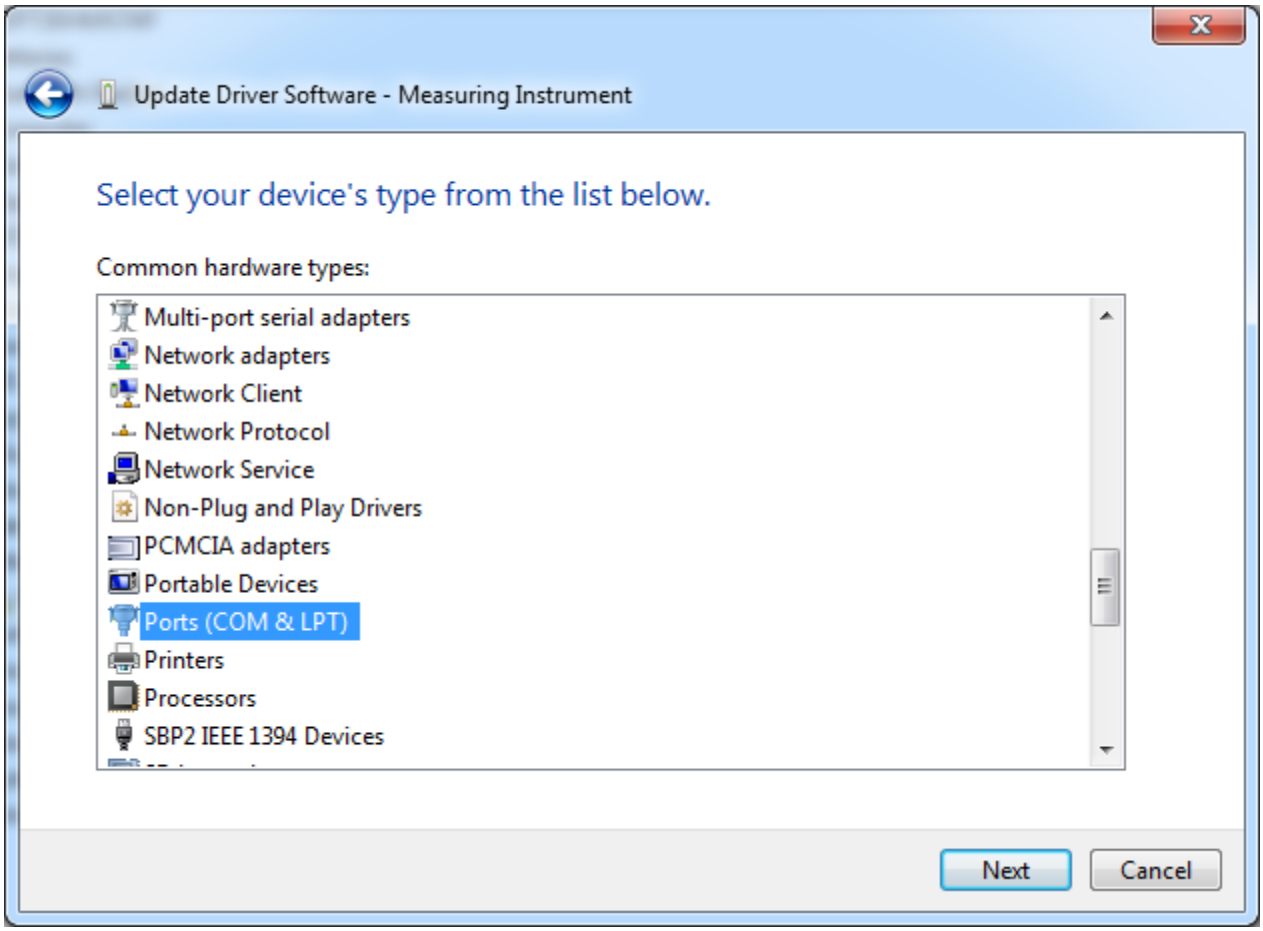
Click “Browse my computer for driver software”.



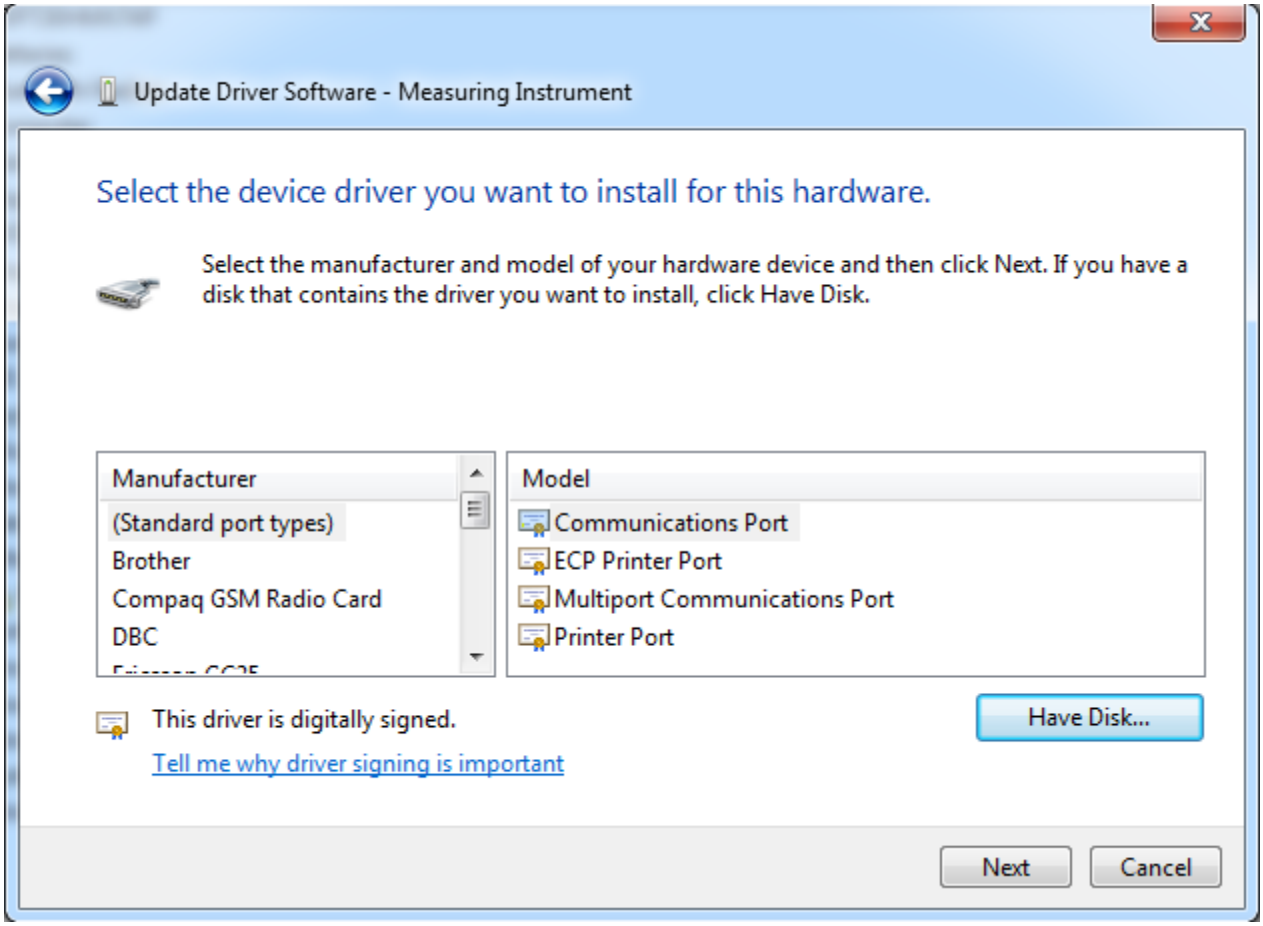
Click "Let me pick from a list of device drivers on my computer"



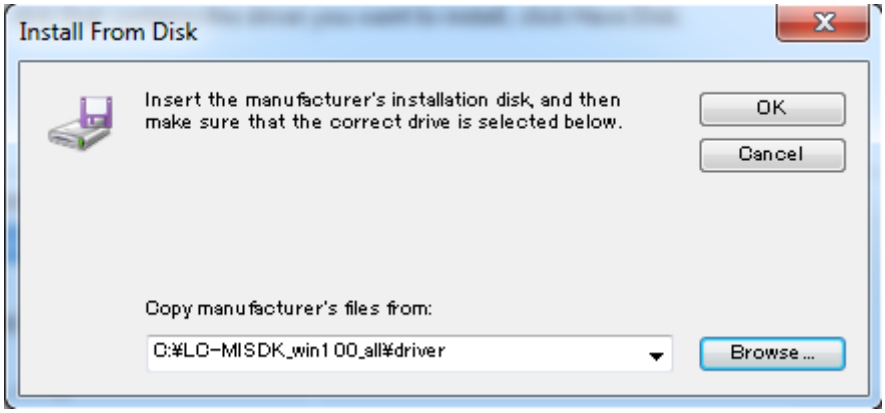
Select “Ports(COM & LPT)” in Common hardware types, click “Next”



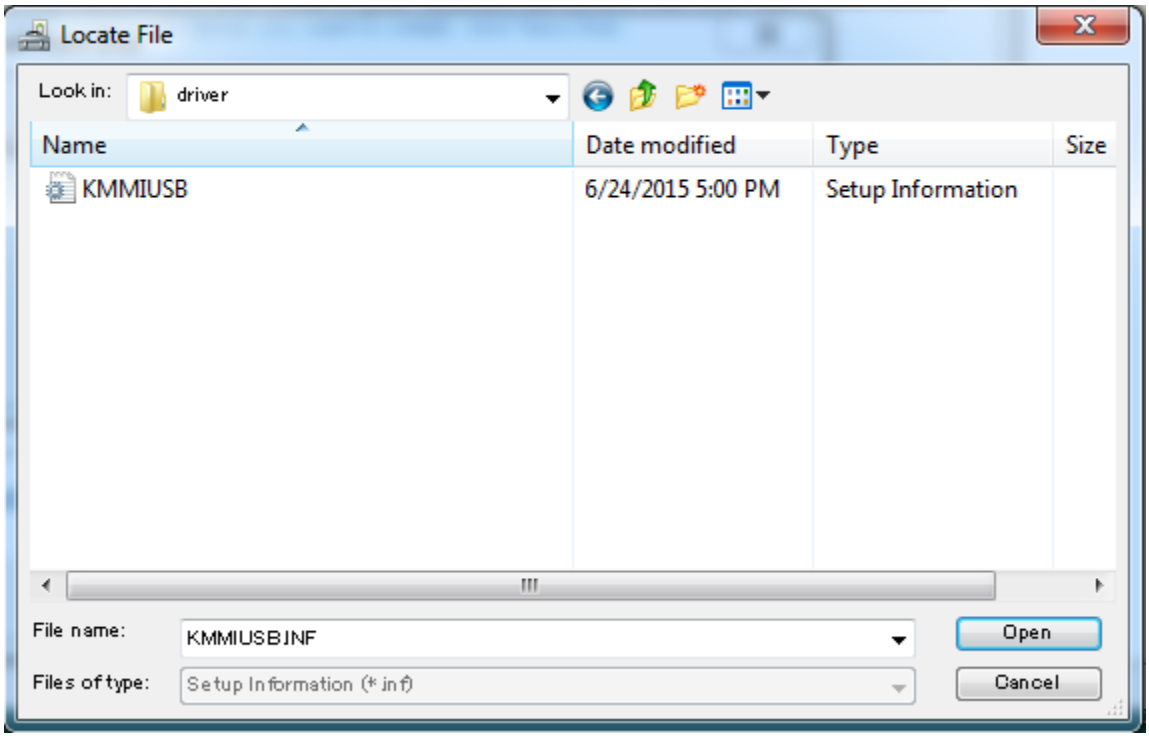
Click “Have Disk…”



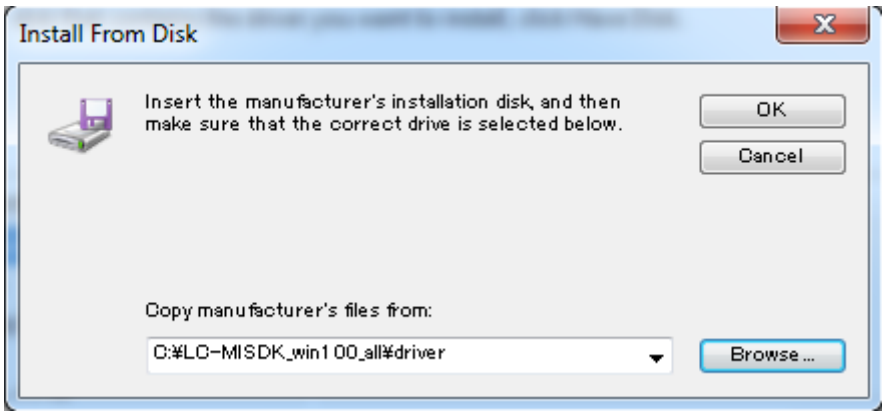
Click “Browse…,”



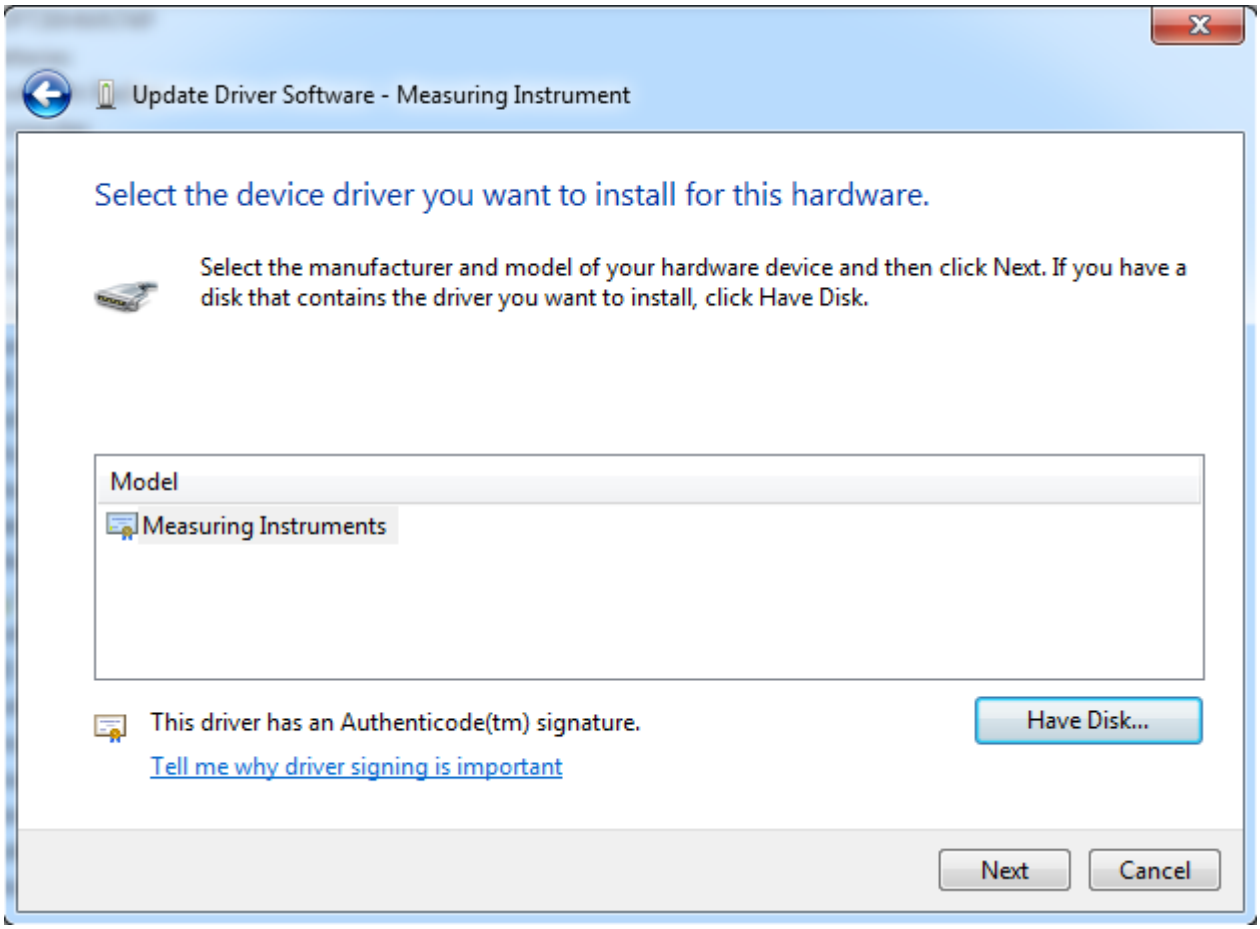
Select “KMMIUSB.INF” in “LC-MISDK_win100_all/driver/”, and click “Open”



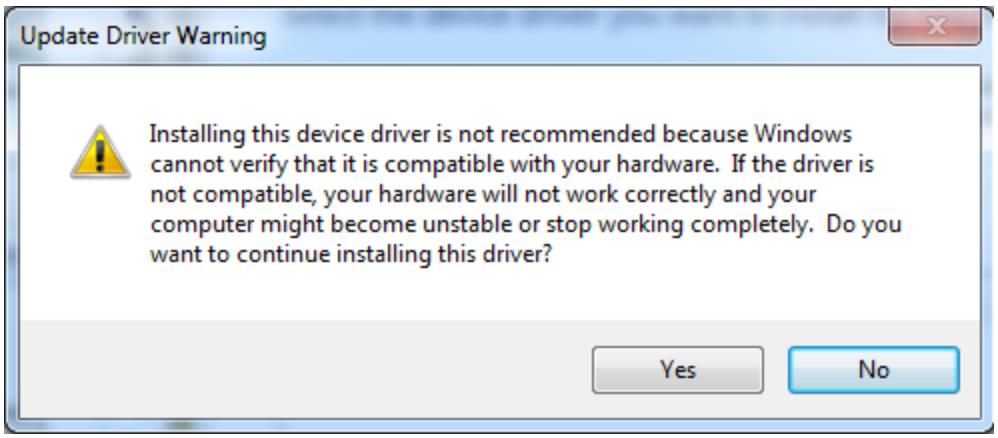
Click “OK”



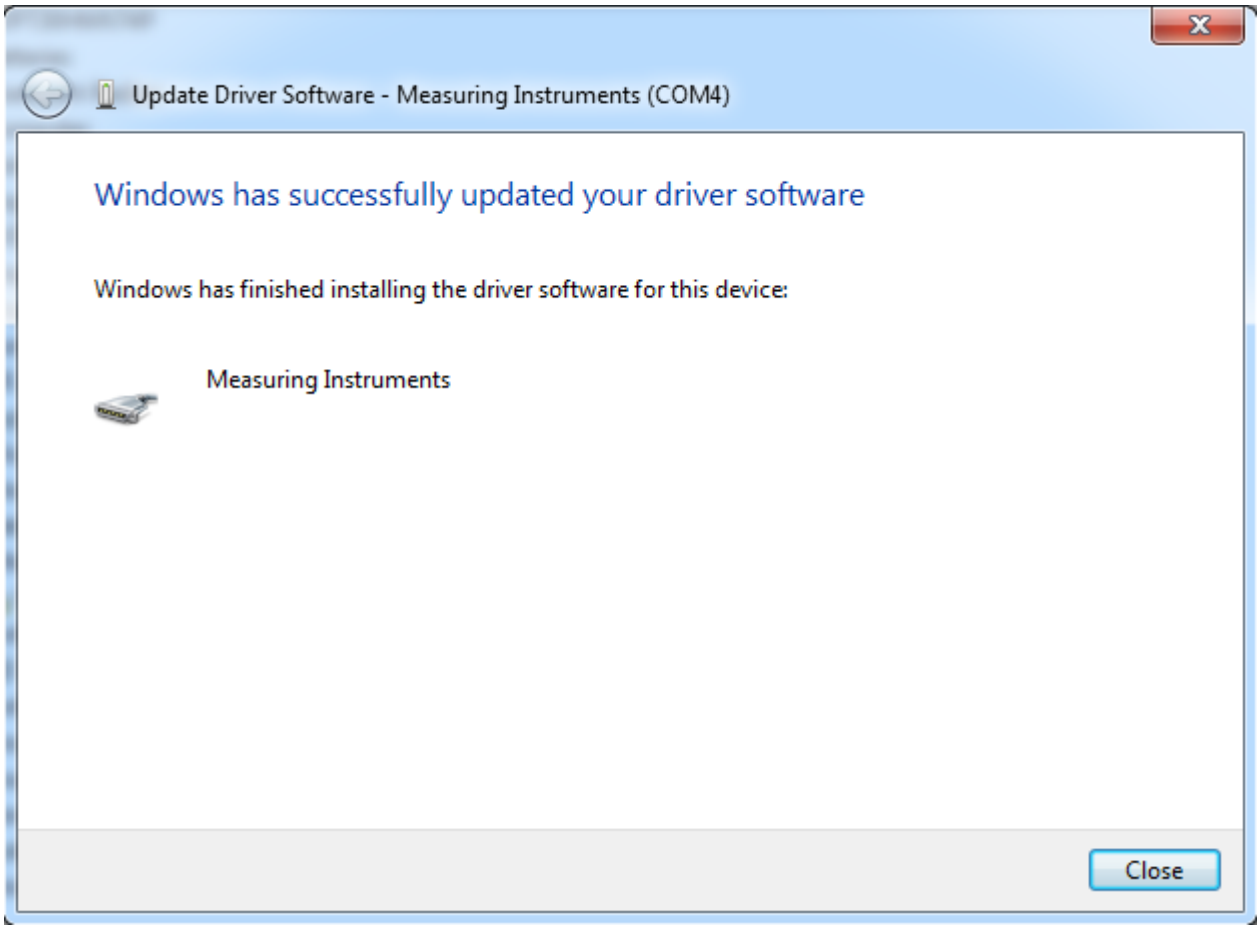
Click “Next”



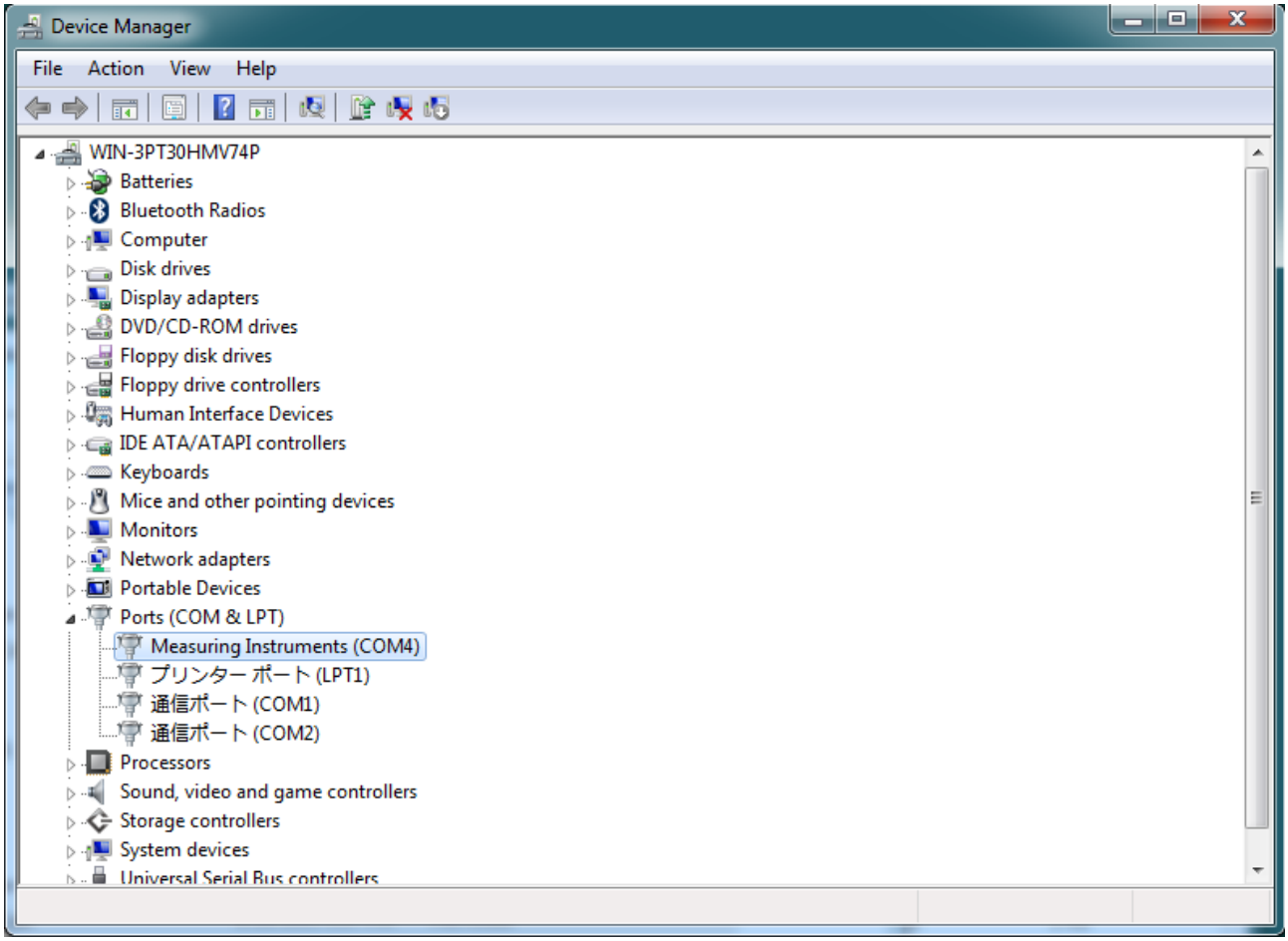
Click “Yes”



Click “Close”


Update Driver Software - Measuring Instruments (COM4)
Windows has successfully updated your driver software
Windows has finished installing the driver software for this device:
Measuring Instruments
Close

Confirm that [Measuring Instruments] does not have the caution icon and has a COM port number. (This COM port number differs in each environment.)



Manual installation is finished here.