LC-MISDK

リファレンスマニュアル

[Version 1.00]

KONICA MINOLTA

●本書で使用しているアプリケーション名などの正式名称

本文中の表記	正式名称
Windows 10	Microsoft® Windows® 10 Pro Operating System
Windows 11	Microsoft® Windows® 11 Pro Operating System

●商標について

●本書に関するご注意

目　次

1. 概要

1.1 はじめに

LC-MISDK は光源色機器用の PC アプリケーションソフトを開発するためのツールです。このマニュアルは LC-MISDK の使用方法を説明します。アプリケーション開発者は Microsoft Visual C#を使用することを想定しており、プログラミングの方法は Microsoft Visual C#で説明しています。

1.2 定義(用語、略語)

用語・略語	意味
カレント測定値	測定器に保存されている最新の測定データ
リモートモード	PC と通信し、各種キーが無効になるモード
接続・接続状態	SDK との通信を開始すること、通信している状態

1.3 動作環境

対応 OS	Windows 10 (32-bit/64-bit) Windows 11
動作環境	.NET Framework 4.5
開発環境　※	Visual Studio 2013 Visual Studio 2015
開発言語	Visual C#
対応プラットフォーム	x64 AnyCPU

※　必要な外部パッケージをインストールするために、インターネット接続環境が必要です。

1.4 対応機種

CS/LS-150 シリーズ	CS-150, CS-160 LS-150, LS-160

2. セットアップ

LC-MISDK を利用するためには、Visual Studio で C#アプリケーションを新規作成した上で LC-MISDK の NuGet パッケージのインストールを実行する必要があります。

		内容
LC-MISDK_win100_all/	nupkg/	NuGet 用パッケージファイル

	SampleProgram/	C#サンプルプログラムのソリューション
	Manual/	リファレンスマニュアル
	driver/	USB 通信ドライバ
	license/	EULA(使用許諾、パッケージライセンス情報)

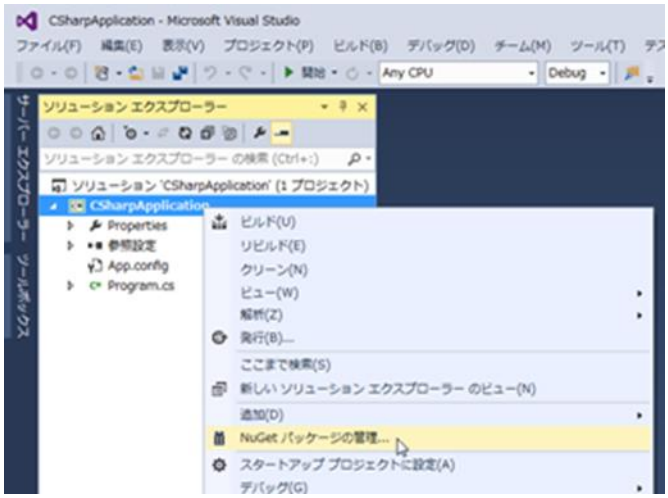
2.1 NuGet パッケージのインストール

LC-MISDK は、Visual Studio2013 以降に搭載されている NuGet というツールを利用してインストールします。NuGet パッケージのインストールによって、アプリケーション開発に必要な DLL や設定ファイルなどがプロジェクトにコピーされ、自動的に参照設定されます。

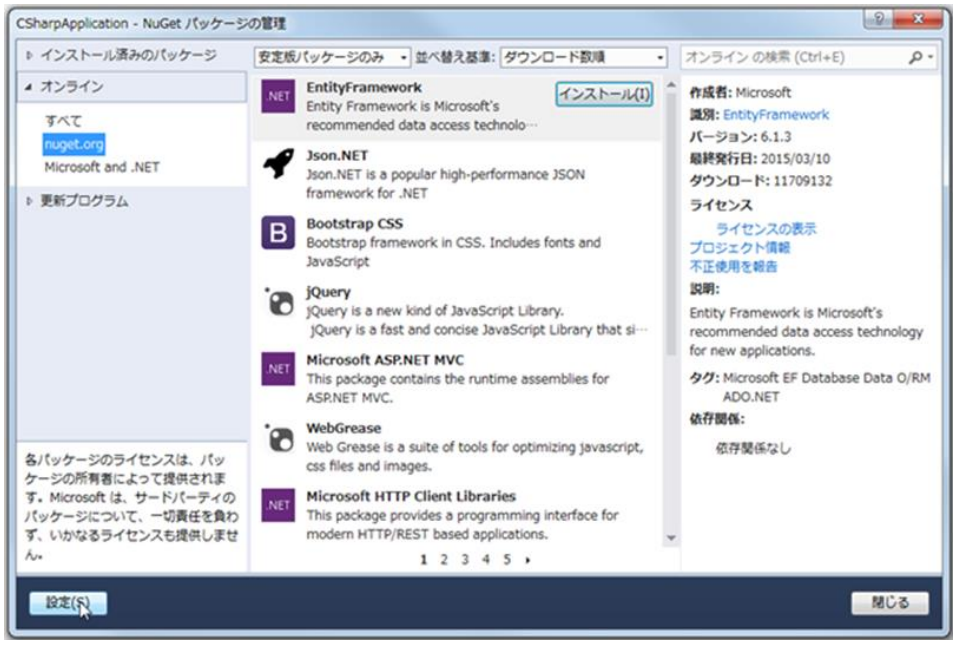
1. インストール手順

<Visual Studio 2013 の場合>

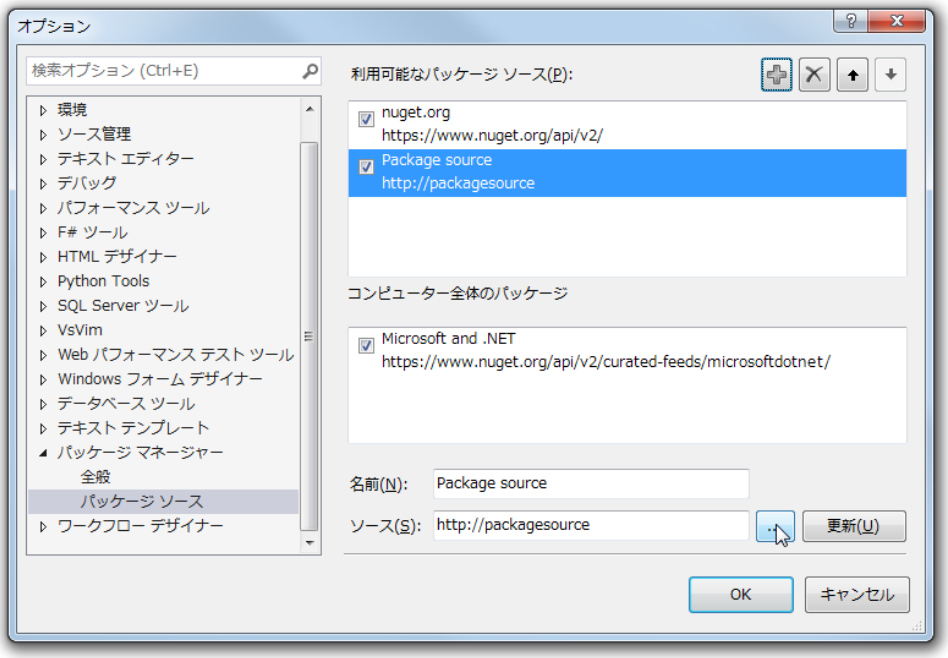
プロジェクトを右クリック、「NuGet パッケージの管理」を選択して、NuGet パッケージ管理メニューを開きます。



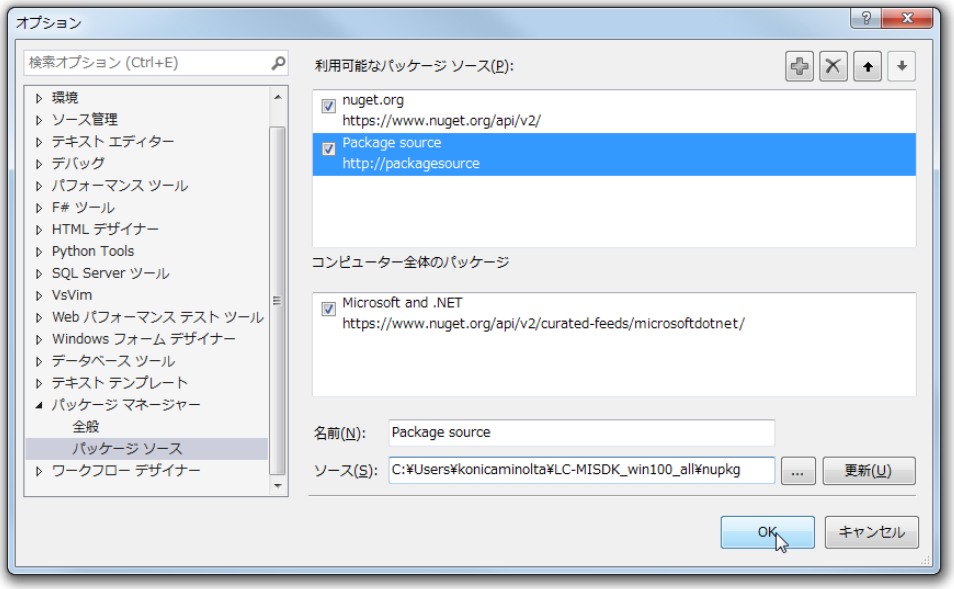
左下の設定ボタンを押してオプションメニューを開きます。



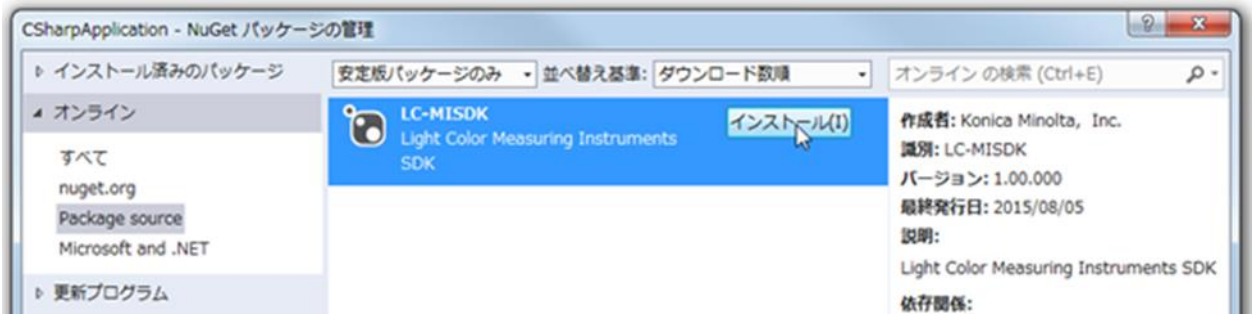
右上の追加ボタンを押してパッケージソースを新規作成します。次に「…」ボタンを押してパッケージソースの場所を指定します。



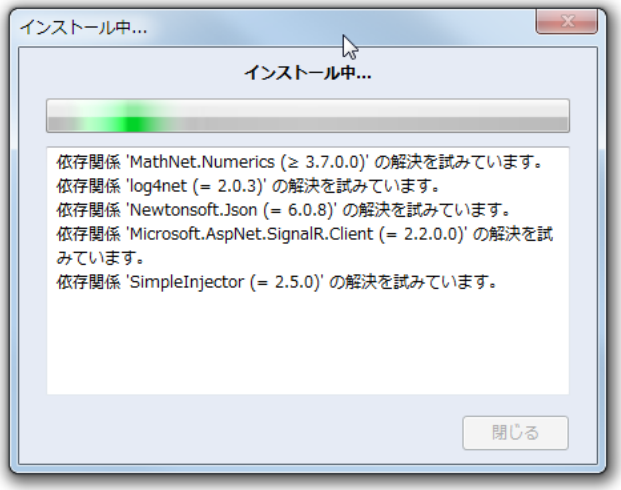
パッケージソースの場所として、CL-MISDK.nupkg が存在するフォルダ（LC-MISDK_win100_all/nupkg/）を指定します。ソースの部分にフォルダが指定されたら「OK」ボタンを押します。



「NuGet パッケージの管理→オンライン」の中に LC-MISDK が表示されるので、選択して「インストール」ボタンを押します。



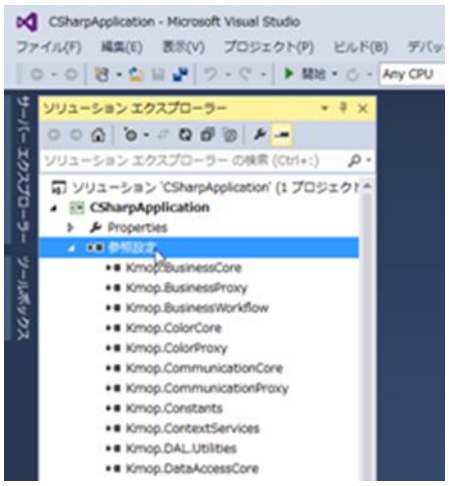
インストールが開始されます。



※　nuget.org から外部パッケージをダウンロードしてインストールするため、インターネット接続環境が必要となります。

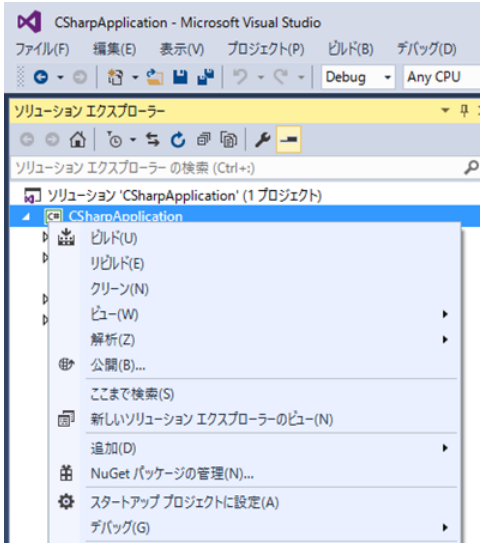
ライセンスの同意確認で「同意する」を選択します。

必要な DLL が参照設定され、インストールは終了です。

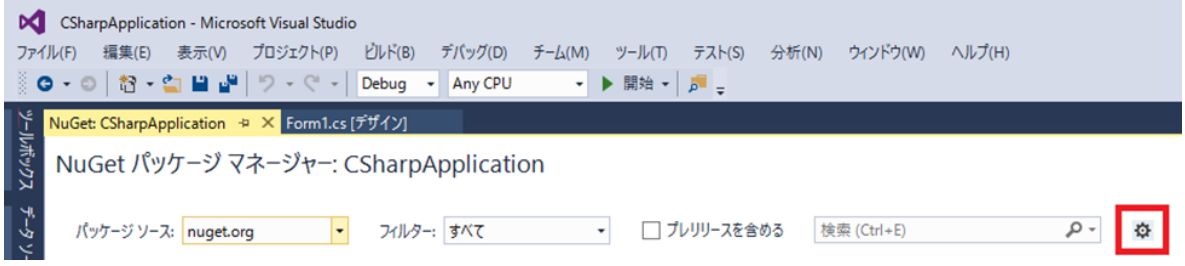


<Visual Studio 2015 の場合>

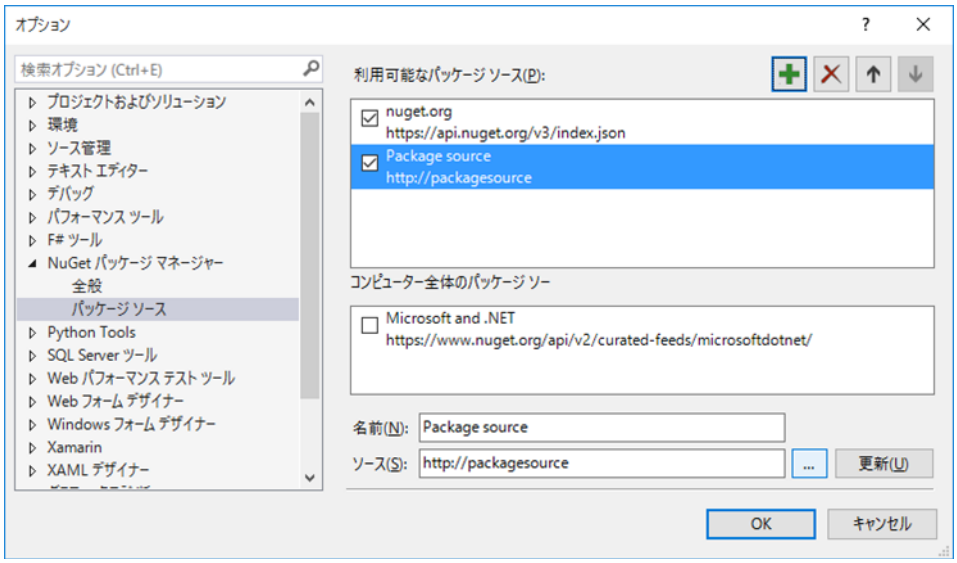
プロジェクトを右クリック、「NuGet パッケージの管理」を選択して、NuGet パッケージマネージャーを開きます。



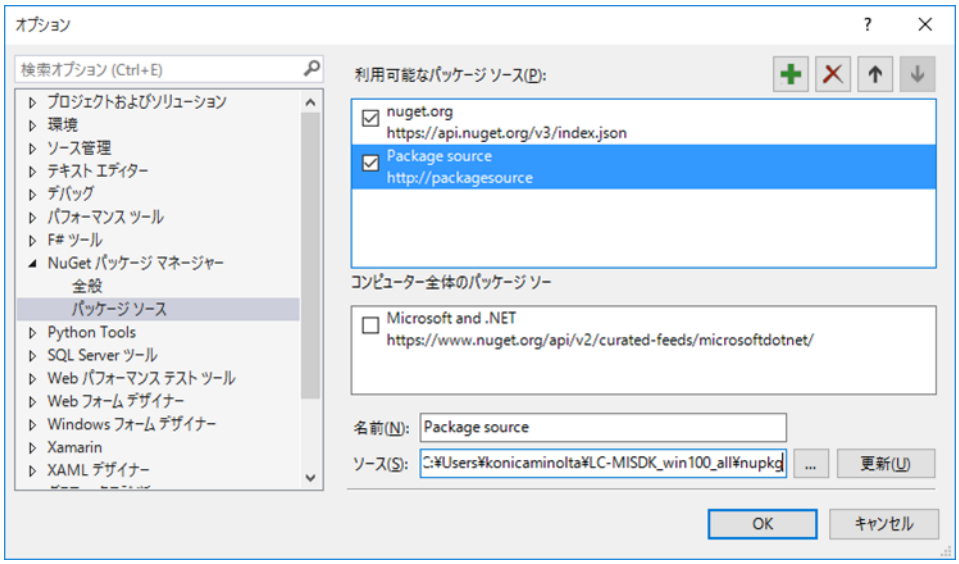
設定アイコンを押してオプションメニューを開きます。



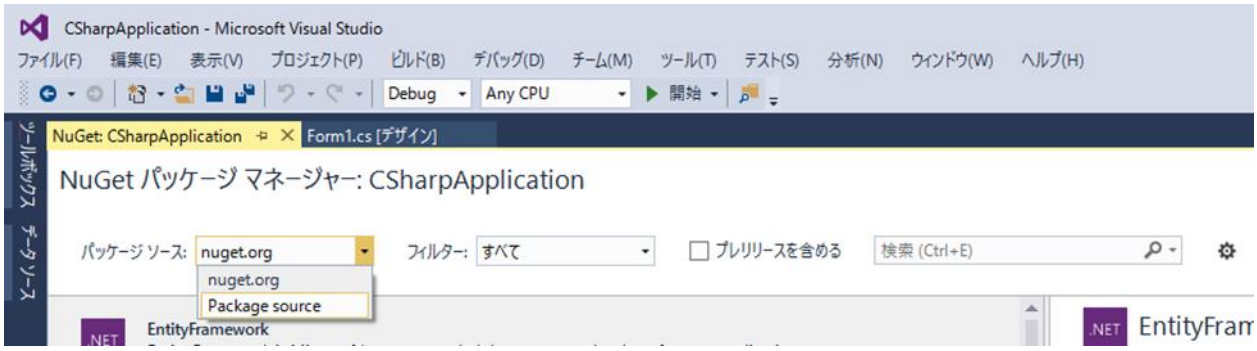
右上の追加ボタンを押してパッケージソースを新規作成します。次に「…」ボタンを押してパッケージソースの場所を指定します。



パッケージソースの場所として、CL-MISDK.nupkg が存在するフォルダ（LC-MISDK_win100_all/nupkg/）を指定します。ソースの部分にフォルダが指定されたら「OK」ボタンを押します。

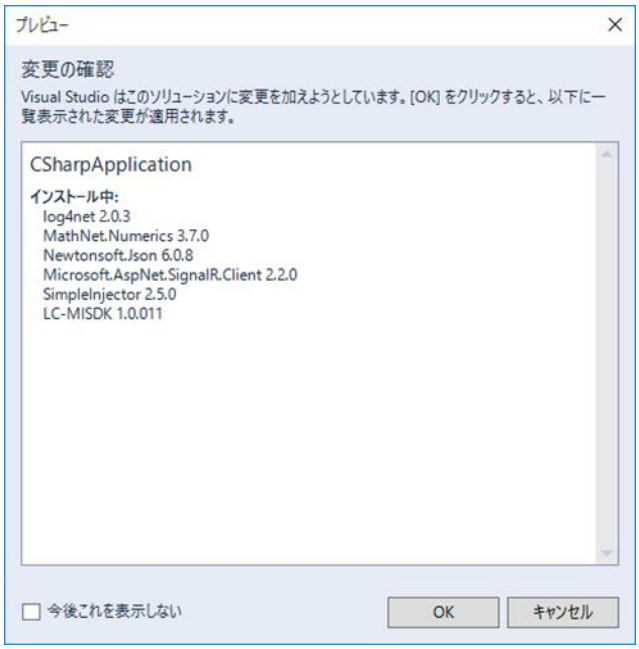


「パッケージソース」から Package source を選択します。



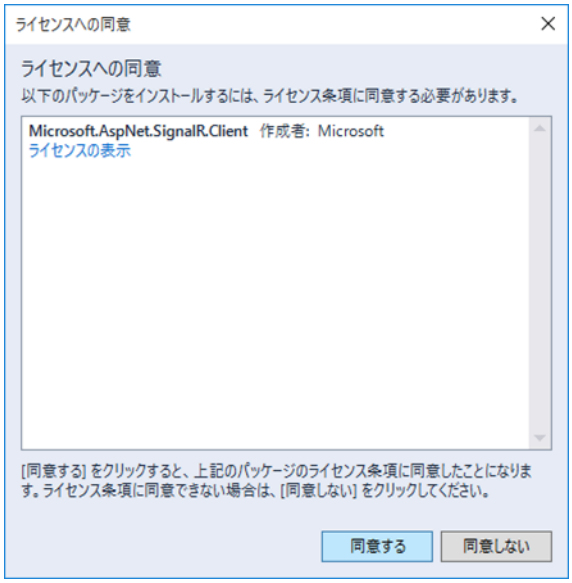
LC-MISDK が表示されるので、選択して「インストール」ボタンを押します。

変更の確認で「OK」を選択します。インストールが開始されます。

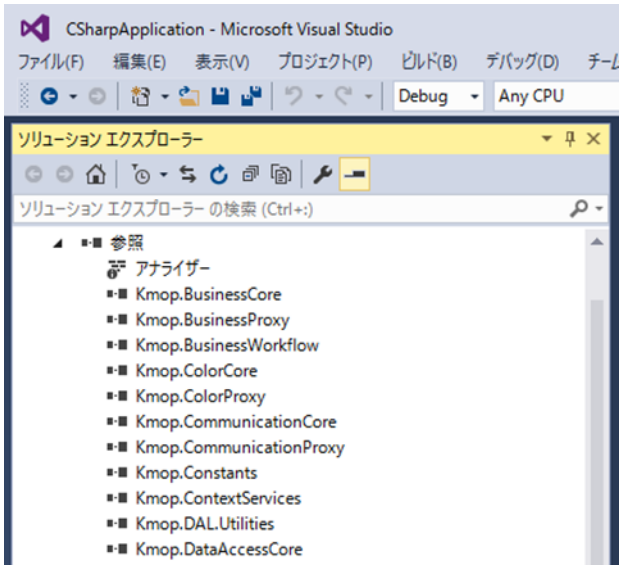


※ nuget.org から外部パッケージをダウンロードしてインストールするため、インターネット接続環境が必要となります。

ライセンスの同意確認で「同意する」を選択します。



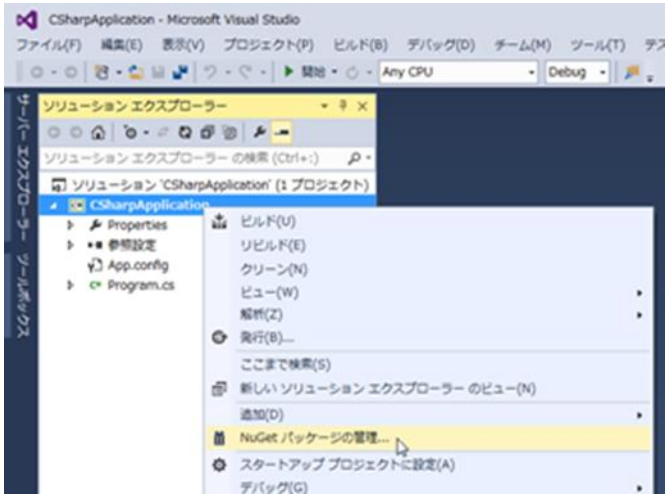
必要な DLL が参照設定され、インストールは終了です。



2. アンインストール

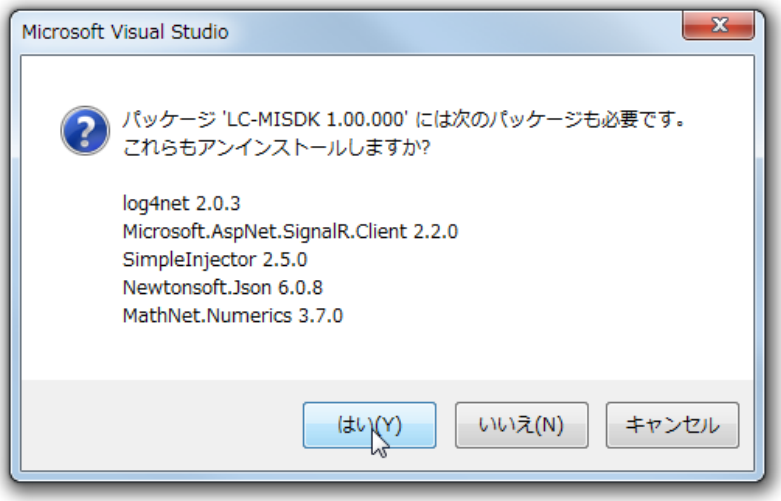
<Visual Studio 2013 の場合>

SDK がインストールされているプロジェクトを右クリック、NuGet パッケージ管理を選択します。



「NuGet パッケージの管理→インストール済のパッケージ」の中に LC-MISDK が表示されるので、「アンインストール」ボタンを押します。

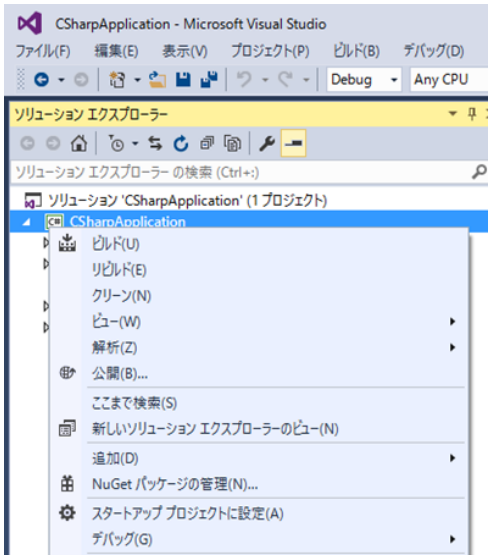
外部パッケージのアンインストールの確認が表示される場合は、「はい」を選択します。



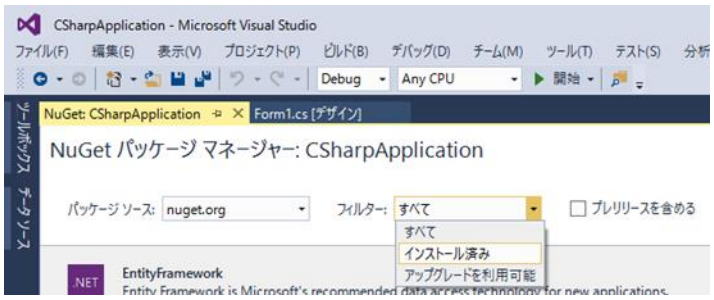
SDK の参照設定の解除と参照ファイルの削除が実行され、アンインストールが完了します。

<Visual Studio 2015 の場合>

SDK がインストールされているプロジェクトを右クリック、NuGet パッケージ管理を選択します。

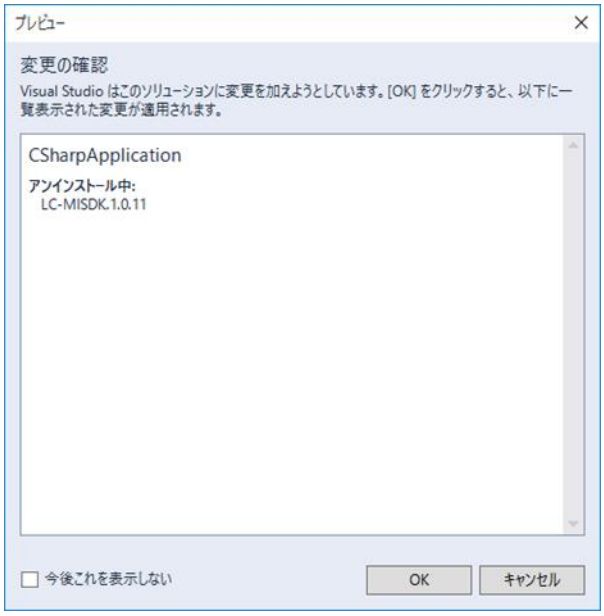


「フィルター」で「インストール済み」を選択します。



LC-MISDK が表示されるので、選択して「アンインストール」ボタンを押します。

変更の確認で「OK」を選択します。



SDK の参照設定の解除と参照ファイルの削除が実行され、アンインストールが完了します。

2.2 基本的な使用方法

LC-MISDK を使用するためには、まず SDK のインスタンスを作成し、そのインスタンスから各 API を呼び出して利用します。SDK のインスタンスは同時に 1 つしか生成できないので、同一の LC-MISDK.dll を参照するアプリケーションを複数同時に起動することはできません。また各機能を利用する前に、接続 API を使用して測定器と接続する必要があります。以下では、この流れを説明します。

1. 名前空間を指定する

```
using Konicaminolta;
```

2. SDK のインスタンスと戻り値クラスを代入するための変数を作成する

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

3. 測定器との通信を開始する

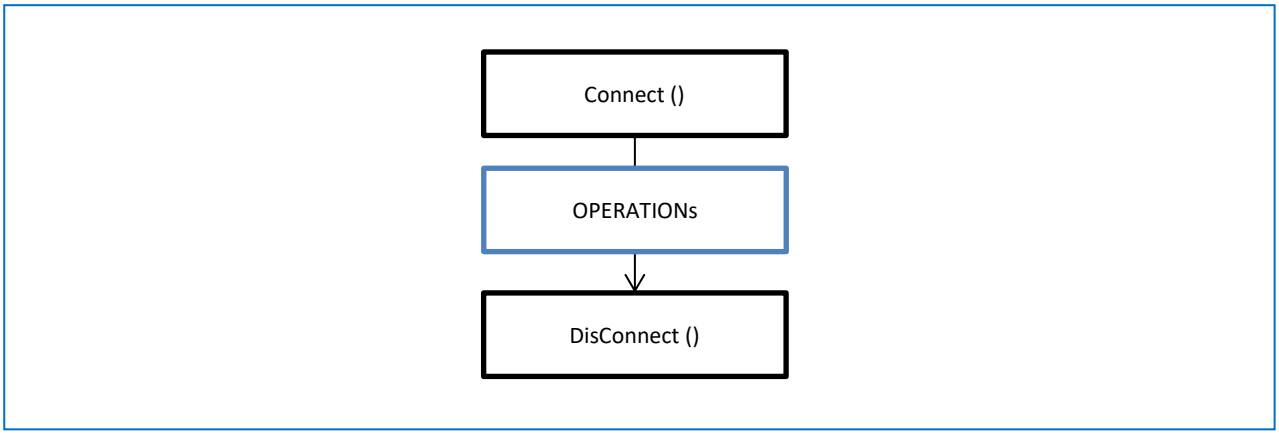
作成したインスタンスを利用して API をコールします。

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

4. 各 API を呼び出して、戻り値のエラー判定を行う

Connect()を使用して測定器へ接続します。この状態で各 API をコールして SDK の機能を実行します。最後に

DisConnect()で測定器との接続を解除します。



各 API の戻り値のエラー処理

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

5. 測定器との通信を終了する

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

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

3.1 概要

CS-150,CS-160 はハンディタイプ色彩輝度計、LS-150,LS-160 はハンディタイプ輝度計です。LC-MISDK でこれらの測定器の本体設定、測定、校正などを PC から実行することができます。CS-150,CS-160 は色彩輝度計、LS-150,LS-160 は輝度計であり、使用できる機能が若干異なります。（3.3、3.4）

3.2 注意事項

測定器を PC に接続している状態で、以下のような操作を行うと、接続 API "Connect()"で測定器に接続できなくなることがあります。この場合は、測定器の電源を OFF、ON してから再度接続してください。

・Windows を起動・再起動・休止状態等にする

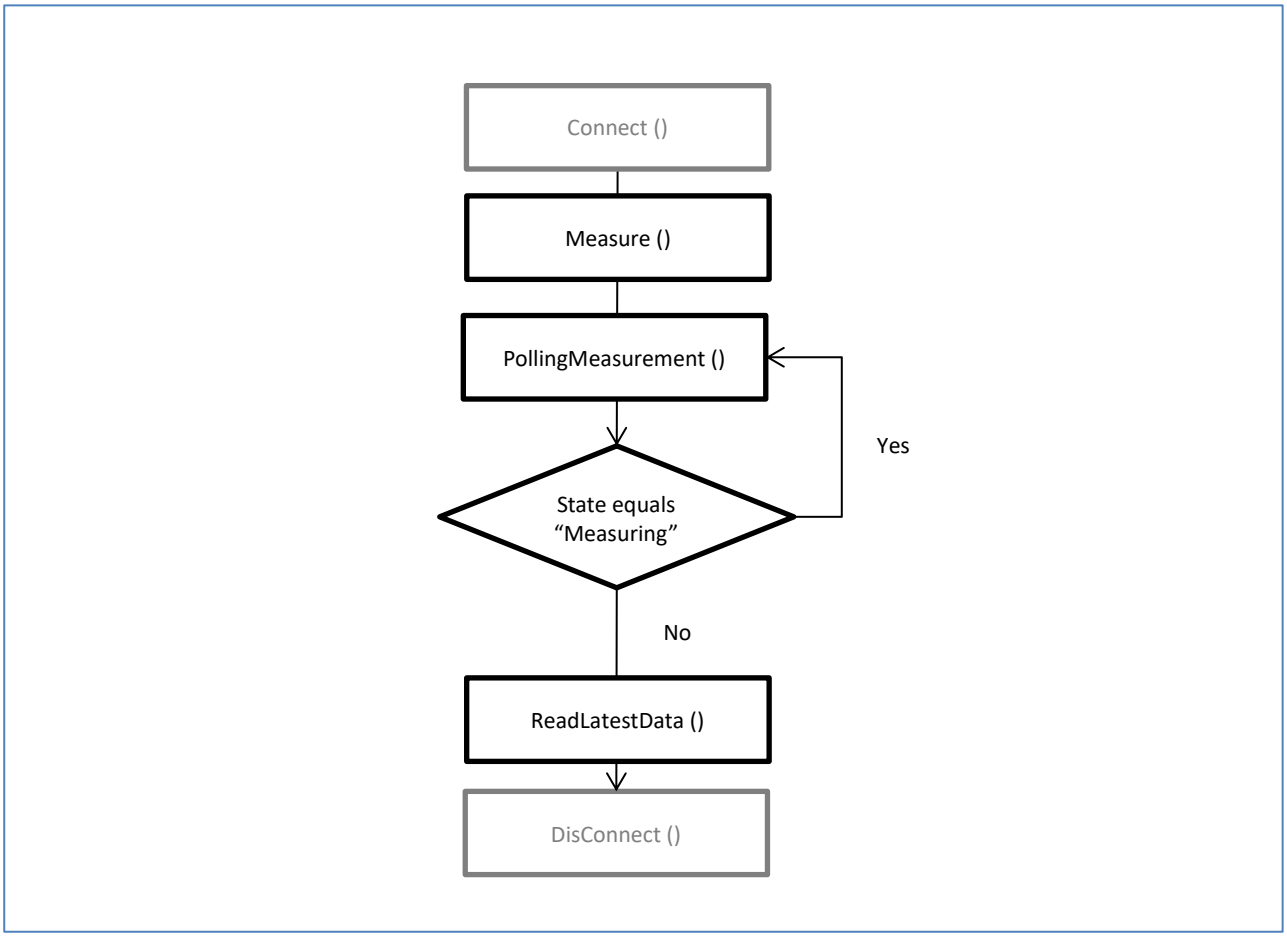
・切断処理を行わずに測定器の電源を OFF にしたり、USB ケーブルを抜いたりする

3.3 ユースケース

<測定>	
1	測定を実行する（測定→ある表色系での測定値を取得）
2	本体に保存されている測定データを読み出す
<設定>	
3	バックライトの設定を変更する
4	測定時間の設定を変更する
<校正>	
5	１点校正を実施する
6	RGB 校正を実施する

1. 測定を実行する

ID	ユースケース名		概要
1	SDK から測定を実行して測定値を取得する	1	測定を開始する
		2	ポーリングして測定終了を待つ
		3	測定値を取得する



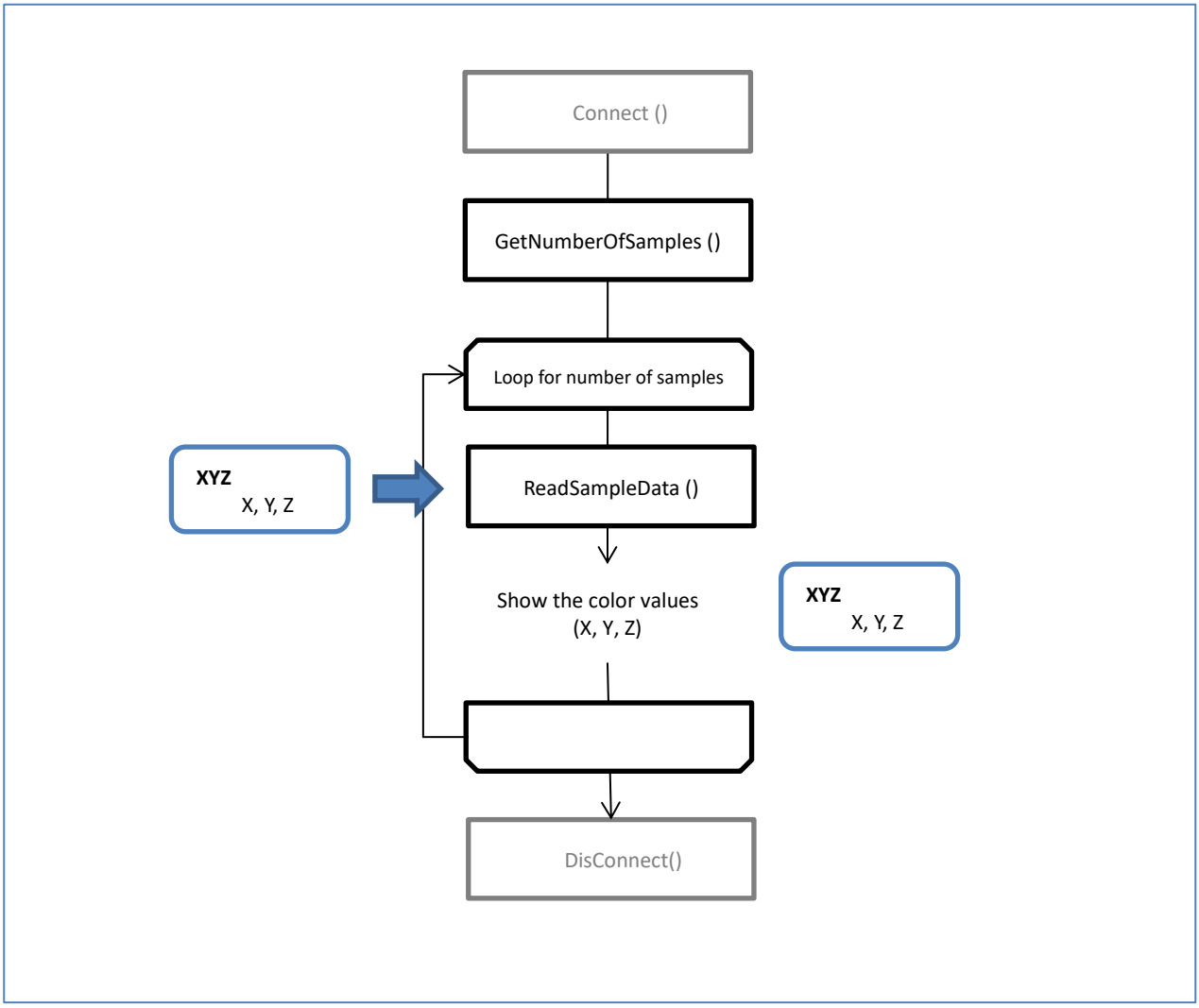
OneShotMeasurementSample.cs

```
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. 本体に保存されている測定データを読み出す

ID	ユースケース名		概要
2	本体に保存されている測定値をある表色系の値として取得する	1	測定器単体で測定を実行する
		2	PC と測定器を接続する
		3	測定器に保存されている測定値の個数を取得する
		4	任意の表色系を指定して測定値を取得する
		5	３で取得した個数分、４を繰り返し実行する



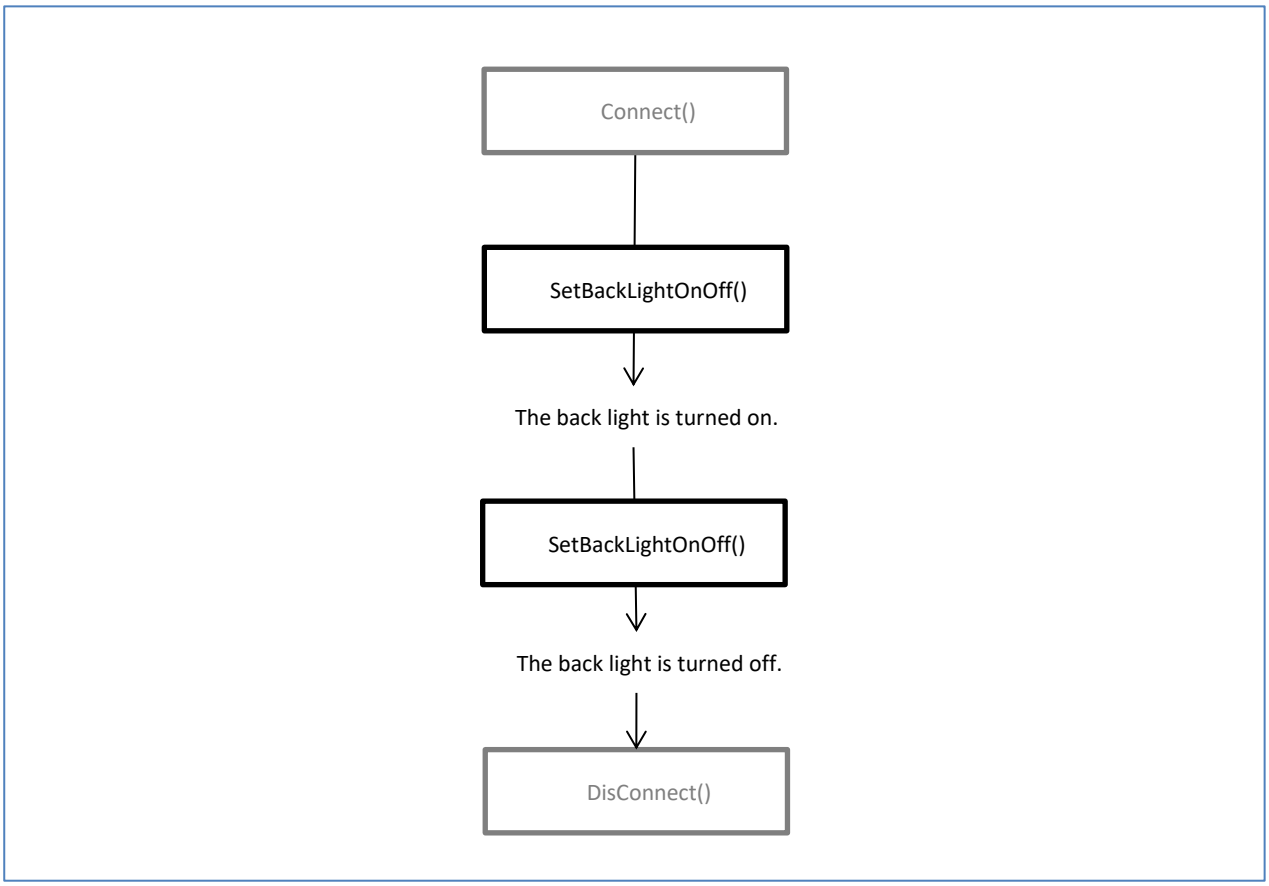
ReadSampleDataSample.cs

```
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. バックライトの設定を変更する

ID	ユースケース名		概要
3	バックライトの設定を変更する	1	バックライトを ON にする
		2	1 秒待つ
		3	バックライトを OFF にする

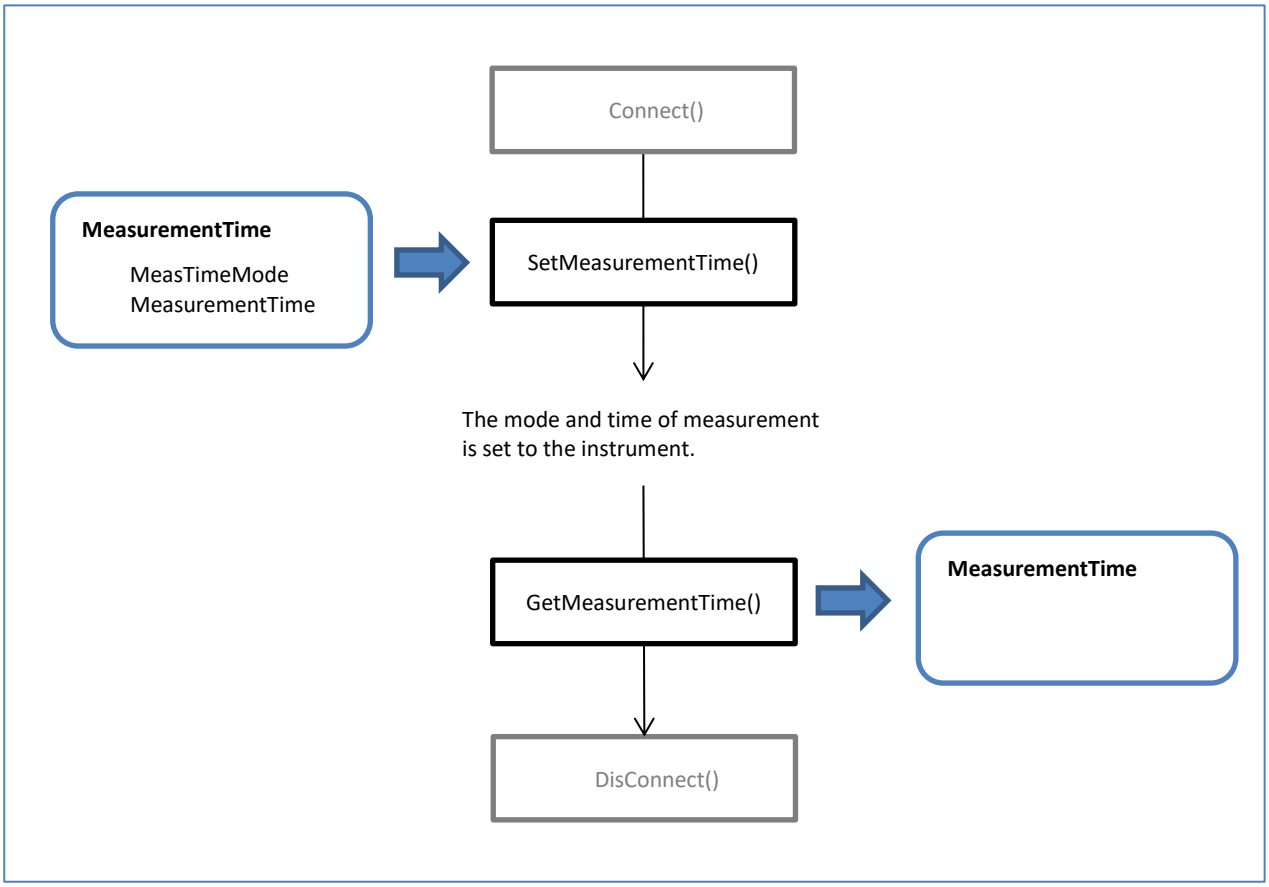


BackLightSettingSample.cs

```
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. 測定時間の設定を変更する

ID	ユースケース名		概要
4	測定時間の設定を変更する	1	MeasurementTime クラスのインスタンスを作成し、同期・非同期の設定を指定する
		2	同期の場合は同期周波数の設定を行う
		3	測定時間の設定を変更する
		4	設定されていることを確認する



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
```

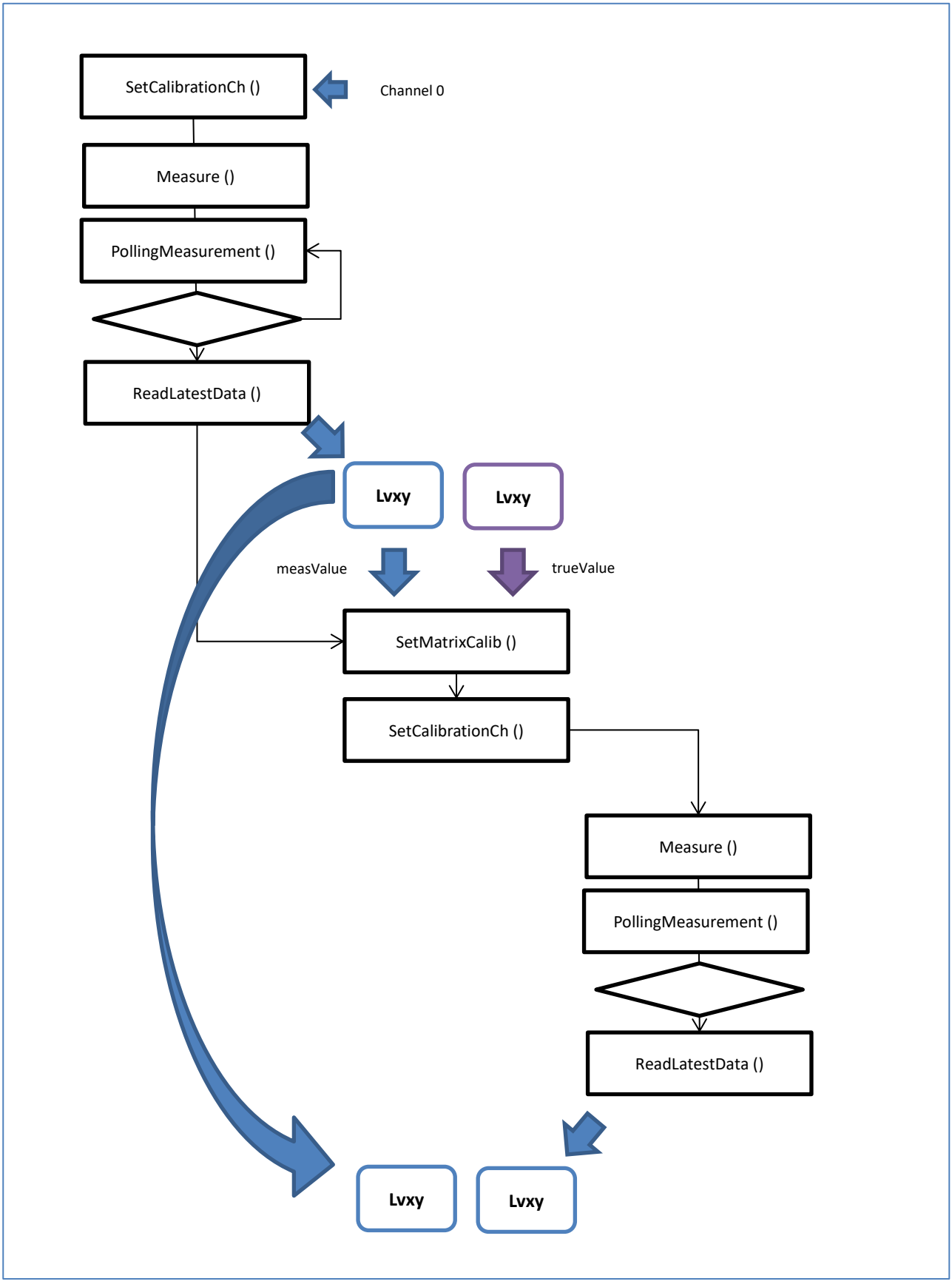
```
            {
                // Display measurement time values
                Console.WriteLine("");
                Console.WriteLine("MeasTimeMode:{0}", inputMeasurementTime.MeasTimeMode);
                Console.WriteLine("ManualMeasurementTime:{0}" ,
inputMeasurementTime.ManualMeasurementTime);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



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

5. 1 点校正を実施する

ユーザー校正は、校正値セットとして、“ある測定値（校正用実測値）”と“その測定値が本来取るべき値（校正用真値）”の２つを指定することで、任意の校正を適用した測定を行える機能です。まず、ユーザー校正係数を設定する API に対して、ユーザー校正の種類、必要な校正値セット、校正係数がセットされる校正チャンネルの３つを指定します。その校正チャンネルを別の API で指定することで、校正が適用された測定を行うことができます。

ID	ユースケース名		概要
6	測定値を元に 1 点校正を実施。その校正値を用いて測定したデータを取得する。	1	校正用チャンネル設定を 0 に指定する ※1
		2	校正前に白色光源を測定する
		3	校正前の測定値を取得して、校正用「実測値」とする
		4	校正用「真値」を作成する
		5	校正タイプを 1 点校正にして、校正係数を測定器に設定する
		6	測定器の校正チャンネル設定を 4 で指定したチャンネルに指定する
		7	校正後に 1 と同じ白色光源を測定する
		8	校正後の測定値を取得する
		9	校正前、校正後の測定値を並べて表示する

※1. 二重に校正されることを防ぐため、校正前の測定値取得は校正係数が設定されないチャンネル（校正チャネル０）を指定して実施してください。


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 user 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 user 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 user 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");
```

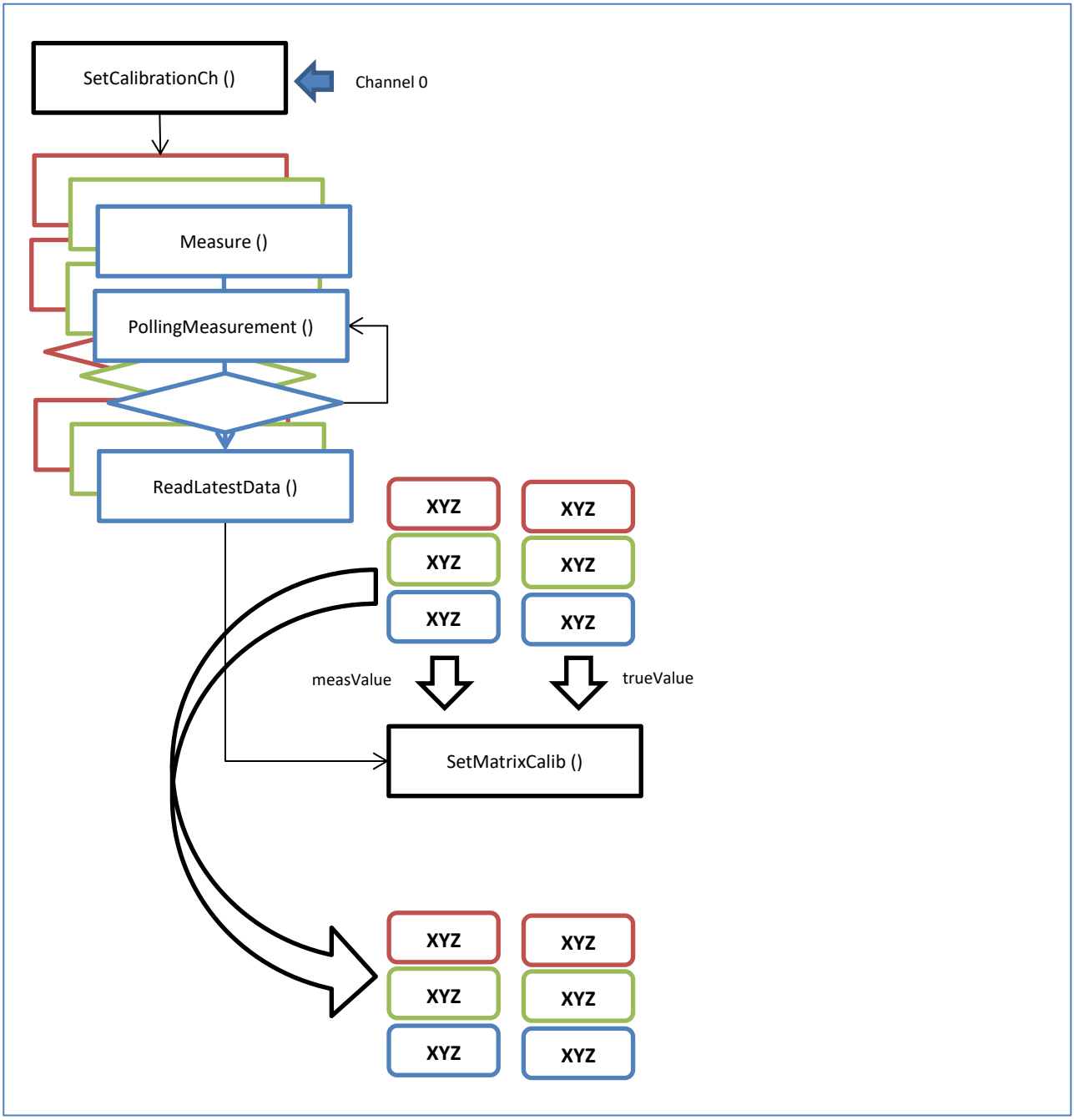
```
            foreach(var beforeCalibValue in beforeValue.ColorSpaceValue){
                Console.WriteLine("{0} : {1}", beforeCalibValue.Key,
beforeCalibValue.Value);
            }
            Console.WriteLine("After user calibration");
            foreach(var afterCalibValue in afterValue.ColorSpaceValue){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

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

6. RGB 校正を実施する

RGB 校正は、1 点校正と同様に校正値セットとして、“ある測定値（校正用実測値）”と“その測定値が本来取るべき値（校正用真値）”を指定することで、任意の校正を適用した測定を行える機能です。1 点校正と異なり、赤、緑、青の 3 種類の光源を測定し、3 種類の校正値セット（校正用実測値と校正用真値）を用意する必要があります。

ID	ユースケース名		概要
6	測定値を元に 1 点校正を実施。その校正値を用いて測定したデータを取得する。	1	校正用チャンネル設定を 0 に指定する ※1
		R1	校正前に赤色光源(R)を測定する
		R2	校正前の測定値を取得して、校正用実測値 “R(X,Y,Z)” とする
		G1	校正前に緑色光源(G)を測定する
		G2	校正前の測定値を取得して、校正用実測値 “G(X,Y,Z)” とする
		B1	校正前に青色光源(B)を測定する
		B2	校正前の測定値を取得して、校正用実測値 “B(X,Y,Z)” とする
		2	校正用真値 ”R(X,Y,Z), G(X,Y,Z), B(X,Y,Z)” を作成する
		3	校正タイプを RGB 校正にして、校正係数を測定器に設定する
		4	校正前の測定値と校正用真値を並べて表示する

※1. 二重に校正されることを防ぐため、校正前の測定値取得は校正係数が設定されないチャンネル（校正チャネル 0 ）を指定して実施してください。


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;
            }

            // Display
            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 対応 API 一覧 (CS-150/CS-160)

Group	API Name	API Description
通信	Connect()	接続します
通信	DisConnect()	切断します
通信	GetDeviceList()	PC に接続されている測定器情報を取得します
測定	Measure()	サンプル測定を 1 度実行します
測定	PollingMeasurement()	測定状態を確認します
測定	CancelMeasurement()	測定を中断します
測定値の取得	ReadLatestData()	最新の測定での各種色彩値データを取得します
測定値の取得	ReadDisplayValue()	表示値の取得
測定データ（本体保存）	GetNumberOfSampleData()	測定器に保存されている測定データ数を取得します
測定データ（本体保存）	ReadSampleData()	測定器から測定データを取得します
測定データ（本体保存）	DeleteSampleData()	測定データを削除します
基準値	SetTargetCh()	使用する基準値チャンネルの設定
基準値	GetTargetCh()	現在指定されている基準値チャンネルの取得
基準値	ReadTargetData()	測定器から基準値を取得します
基準値	DeleteTargetData()	基準値を削除します
基準値	WriteTargetData()	測定器に利用する基準値データを書き込みます
測定時間	SetMeasurementTime()	測定時間を設定します
測定時間	GetMeasurementTime()	測定時間を取得します
同期	SetSyncMode()	同期測定の設定
同期	GetSyncMode()	同期測定設定の取得
Peak/Valley	SetPeakValley()	Peak/Valley の設定をします
Peak/Valley	GetPeakValley()	Peak/Valley の設定を取得します
レンズ設定	SetCloseUpLens()	クローズアップレンズを設定します
レンズ設定	GetCloseUpLens()	クローズアップレンズ設定を取得します
ユーザー校正チャネル	SetCalibrationCh()	使用するユーザー校正チャンネルの設定
ユーザー校正チャネル	GetCalibrationCh()	現在指定されているユーザー校正チャンネルの取得
ユーザー校正	SetMatrixCalib()	ユーザー校正係数の設定をします
ユーザー校正	GetCalibData()	ユーザー校正係数の取得をします
ユーザー校正	DeleteCalibData()	ユーザー校正係数を削除します
電源設定	SetAutoPowerOff()	オートパワーオフ設定を設定します
電源設定	GetAutoPowerOff()	オートパワーオフ設定を取得します

バックライト	SetBackLightOnOff()	バックライトを設定します
バックライト	GetBackLightOnOff()	バックライトの設定を取得します
バックライト	SetBackLightLevel()	バックライトの明るさを設定します
バックライト	GetBackLightLevel()	バックライトの明るさ設定を取得します
表示桁数	SetColorDispDigit()	色度表示桁数の設定をします
表示桁数	GetColorDispDigit()	色度表示桁数の設定を取得します
表示形式 （絶対値、差分、比率）	SetDisplayType()	表示形式を設定します
表示形式 （絶対値、差分、比率）	GetDisplayType()	表示形式設定を取得します
言語・日時	SetDisplayLanguage()	言語を設定します
言語・日時	GetDisplayLanguage()	言語設定を取得します
言語・日時	SetDateTime()	日時設定を格納します
言語・日時	GetDateTime()	日時設定を取得します
言語・日時	SetDateFormat()	日時フォーマットを設定します
言語・日時	GetDateFormat()	日時フォーマットを取得します
表色モード本体表示	SetColorModeDisplayOnOff()	各表色モードの On/Off を設定します。
表色モード本体表示	GetColorModeDisplayOnOff()	各表色モードの On/Off の設定を取得します
表色モード本体表示	SetColorMode()	表色モードを設定を指定します
表色モード本体表示	GetColorMode()	表色モードを設定を取得します
保存設定	SetDataSaveMode()	データ保存方法の設定します
保存設定	GetDataSaveMode()	データ保存方法の設定を取得します
定期校正設定	SetPeriodicCalibNotify()	定期校正メッセージの表示を設定します
定期校正設定	GetPeriodicCalibNotify()	定期校正メッセージの表示を取得します
ボタン設定	SetToggleOnOff()	測定器の測定ボタンのトグル設定をします
ボタン設定	GetToggleOnOff()	測定器の測定ボタンのトグル設定を取得します
ボタン設定	SetTriggerOnOff()	測定器の測定ボタンの有効無効状態を設定します
ボタン設定	GetTriggerOnOff()	測定器の測定ボタンの有効無効状態を取得します
本体・SDK 情報取得	GetDeviceInfo()	機器情報を取得します
本体・SDK 情報取得	GetSDKVersion()	SDK のバージョンを取得します
単位設定	GetLuminanceUnit()	輝度単位設定の取得をします

3.5 対応 API 一覧 (LS-150/LS-160)

Group	API Name	API Description
通信	Connect()	接続します
通信	DisConnect()	切断します
通信	GetDeviceList()	PC に接続されている測定器情報を取得します
測定	Measure()	サンプル測定を 1 度実行します
測定	PollingMeasurement()	測定状態を確認します
測定	CancelMeasurement()	測定を中断します
測定値の取得	ReadLatestData()	最新の測定での各種色彩値データを取得します
測定値の取得	ReadDisplayValue()	表示値の取得
測定データ（本体保存）	GetNumberOfSampleData()	測定器に保存されている測定データ数を取得します
測定データ（本体保存）	ReadSampleData()	測定器から測定データを取得します
測定データ（本体保存）	DeleteSampleData()	測定データを削除します
基準値	SetTargetCh()	使用する基準値チャンネルの設定
基準値	GetTargetCh()	現在指定されている基準値チャンネルの取得
基準値	ReadTargetData()	測定器から基準値を取得します
基準値	DeleteTargetData()	基準値を削除します
基準値	WriteTargetData()	測定器に利用する基準値データを書き込みます
測定時間	SetMeasurementTime()	測定時間を設定します
測定時間	GetMeasurementTime()	測定時間を取得します
同期	SetSyncMode()	同期測定の設定
同期	GetSyncMode()	同期測定設定の取得
Peak/Valley	SetPeakValley()	Peak/Valley の設定をします
Peak/Valley	GetPeakValley()	Peak/Valley の設定を取得します
レンズ設定	SetCloseUpLens()	クローズアップレンズを設定します
レンズ設定	GetCloseUpLens()	クローズアップレンズ設定を取得します
ユーザー校正チャネル	SetCalibrationCh()	使用するユーザー校正チャンネルの設定
ユーザー校正チャネル	GetCalibrationCh()	現在指定されているユーザー校正チャンネルの取得
ユーザー校正	SetMatrixCalib()	ユーザー校正係数の設定をします
ユーザー校正	GetCalibData()	ユーザー校正係数の取得をします
ユーザー校正	DeleteCalibData()	ユーザー校正係数を削除します
CCF	SetCCF()	CCF（色補正係数）設定を指定します
CCF	GetCCF()	CCF（色補正係数）設定を取得します

電源設定	SetAutoPowerOff()	オートパワーオフ設定を設定します
電源設定	GetAutoPowerOff()	オートパワーオフ設定を取得します
バックライト	SetBackLightOnOff()	バックライトを設定します
バックライト	GetBackLightOnOff()	バックライトの設定を取得します
バックライト	SetBackLightLevel()	バックライトの明るさを設定します
バックライト	GetBackLightLevel()	バックライトの明るさ設定を取得します
表示形式 （絶対値、差分、比率）	SetDisplayType()	表示形式を設定します
表示形式 （絶対値、差分、比率）	GetDisplayType()	表示形式設定を取得します
言語・日時	SetDisplayLanguage()	言語を設定します
言語・日時	GetDisplayLanguage()	言語設定を取得します
言語・日時	SetDateTime()	日時設定を格納します
言語・日時	GetDateTime()	日時設定を取得します
言語・日時	SetDateFormat()	日時フォーマットを設定します
言語・日時	GetDateFormat()	日時フォーマットを取得します
保存設定	SetDataSaveMode()	データ保存方法の設定します
保存設定	GetDataSaveMode()	データ保存方法の設定を取得します
定期校正設定	SetPeriodicCalibNotify()	定期校正メッセージの表示を設定します
定期校正設定	GetPeriodicCalibNotify()	定期校正メッセージの表示を取得します
ボタン設定	SetToggleOnOff()	測定器の測定ボタンのトグル設定をします
ボタン設定	GetToggleOnOff()	測定器の測定ボタンのトグル設定を取得します
ボタン設定	SetTriggerOnOff()	測定器の測定ボタンの有効無効状態を設定します
ボタン設定	GetTriggerOnOff()	測定器の測定ボタンの有効無効状態を取得します
本体・SDK 情報取得	GetDeviceInfo()	機器情報を取得します
本体・SDK 情報取得	GetSDKVersion()	SDK のバージョンを取得します
単位設定	GetLuminanceUnit()	輝度単位設定の取得をします

3.6 対応表色系

	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 仕様

4.1 API 一覧

Group	API Name	API Description
通信	Connect()	接続します
通信	DisConnect()	切断します
通信	GetDeviceList()	PC に接続されている測定器情報を取得します
測定	Measure()	サンプル測定を 1 度実行します
測定	PollingMeasurement()	測定状態を確認します
測定	CancelMeasurement()	測定を中断します
測定値の取得	ReadLatestData()	最新の測定での各種色彩値データを取得します
測定値の取得	ReadDisplayValue()	表示値の取得
測定データ（本体保存）	GetNumberOfSampleData()	測定器に保存されている測定データ数を取得します
測定データ（本体保存）	ReadSampleData()	測定器から測定データを取得します
測定データ（本体保存）	DeleteSampleData()	測定データを削除します
基準値	SetTargetCh()	使用する基準値チャンネルの設定
基準値	GetTargetCh()	現在指定されている基準値チャンネルの取得
基準値	ReadTargetData()	測定器から基準値を取得します
基準値	DeleteTargetData()	基準値を削除します
基準値	WriteTargetData()	測定器に利用する基準値データを書き込みます
測定時間	SetMeasurementTime()	測定時間を設定します
測定時間	GetMeasurementTime()	測定時間を取得します
同期	SetSyncMode()	同期測定の設定
同期	GetSyncMode()	同期測定設定の取得
Peak/Valley	SetPeakValley()	Peak/Valley の設定をします
Peak/Valley	GetPeakValley()	Peak/Valley の設定を取得します
レンズ設定	SetCloseUpLens()	クローズアップレンズを設定します
レンズ設定	GetCloseUpLens()	クローズアップレンズ設定を取得します
ユーザー校正チャネル	SetCalibrationCh()	使用するユーザー校正チャンネルの設定
ユーザー校正チャネル	GetCalibrationCh()	現在指定されているユーザー校正チャンネルの取得
ユーザー校正	SetMatrixCalib()	ユーザー校正係数の設定をします
ユーザー校正	GetCalibData()	ユーザー校正係数の取得をします
ユーザー校正	DeleteCalibData()	ユーザー校正係数を削除します
CCF	SetCCF()	CCF（色補正係数）設定を指定します
CCF	GetCCF()	CCF（色補正係数）設定を取得します

電源設定	SetAutoPowerOff()	オートパワーオフ設定を設定します
電源設定	GetAutoPowerOff()	オートパワーオフ設定を取得します
バックライト	SetBackLightOnOff()	バックライトを設定します
バックライト	GetBackLightOnOff()	バックライトの設定を取得します
バックライト	SetBackLightLevel()	バックライトの明るさを設定します
バックライト	GetBackLightLevel()	バックライトの明るさ設定を取得します
表示桁数	SetColorDispDigit()	色度表示桁数の設定をします
表示桁数	GetColorDispDigit()	色度表示桁数の設定を取得します
表示形式 （絶対値、差分、比率）	SetDisplayType()	表示形式を設定します
表示形式 （絶対値、差分、比率）	GetDisplayType()	表示形式設定を取得します
言語・日時	SetDisplayLanguage()	言語を設定します
言語・日時	GetDisplayLanguage()	言語設定を取得します
言語・日時	SetDateTime()	日時設定を格納します
言語・日時	GetDateTime()	日時設定を取得します
言語・日時	SetDateFormat()	日時フォーマットを設定します
言語・日時	GetDateFormat()	日時フォーマットを取得します
表色モード本体表示	SetColorModeDisplayOnOff()	各表色モードの On/Off を設定します。
表色モード本体表示	GetColorModeDisplayOnOff()	各表色モードの On/Off の設定を取得します
表色モード本体表示	SetColorMode()	表色モードを設定を指定します
表色モード本体表示	GetColorMode()	表色モードを設定を取得します
保存設定	SetDataSaveMode()	データ保存方法の設定します
保存設定	GetDataSaveMode()	データ保存方法の設定を取得します
定期校正設定	SetPeriodicCalibNotify()	定期校正メッセージの表示を設定します
定期校正設定	GetPeriodicCalibNotify()	定期校正メッセージの表示を取得します
ボタン設定	SetToggleOnOff()	測定器の測定ボタンのトグル設定をします
ボタン設定	GetToggleOnOff()	測定器の測定ボタンのトグル設定を取得します
ボタン設定	SetTriggerOnOff()	測定器の測定ボタンの有効無効状態を設定します
ボタン設定	GetTriggerOnOff()	測定器の測定ボタンの有効無効状態を取得します
本体・SDK 情報取得	GetDeviceInfo()	機器情報を取得します
本体・SDK 情報取得	GetSDKVersion()	SDK のバージョンを取得します
単位設定	GetLuminanceUnit()	輝度単位設定の取得をします

4.2 API のフォーマット

API 仕様書のフォーマット

各 API は以下のフォーマットで記載されています。

概要：

関数で実行できる処理について説明しています

形式：

関数の書式について説明しています

引数：

関数の引数について説明をしています

戻り値：

関数を利用した際に返ってくる戻り値について説明をしています

戻り値の種別として以下の 3 種類があります。

種別	
正常	処理に成功した際に返ってきます
エラー	処理に失敗した際に返ってきます
警告	処理には成功はしたが、ユーザーに対して情報を提供する必要がある際に返ってきます

説明：

関数を利用する際に必要な情報等や注意事項などを説明しています

Class:　ReturnMessage

すべての API は以下の戻り値クラスを返します。

概要：

各 API の戻り値に用いるクラス

形式：

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

変数：

変数	説明
errorCode	エラーコード
errorMessage	エラーメッセージ（英語）

説明：

エラーコードには、要約されたエラーメッセージが格納されます。

エラーメッセージには、詳細なエラーの説明がが格納されます。

表色系クラス

LC-MISDK では、各 API を使用して任意の表色系における測定値を取得したり、任意の表色系における基準値を設定したりすることができます。ある表色系での測定値の取得もしくは設定を実行するには、各表色系クラスのインスタンスを作成してこれらの API の引数に指定します。※ただし、使用できる表色系は各機種によって異なります。

例えば、XYZ で測定値を取得したい場合は、XYZ クラスのインスタンスを作成し、測定値取得 API の引数に指定します。

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

XYZ 測定値に含まれる各色彩値 X, Y, Z には、以下のようにアクセスできます。

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

また、Lvxy で基準値を取得したい場合は、以下のようにします。Lvxy では、コンストラクタで輝度単位を指定することもできます。

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

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

sdk.WriteTargetData(lvxyValue);
```

さらに詳しい使い方については、表色系クラスを参照してください。

4.3 通信

Connect()

概要：

指定した仮想 COM ポートに繋がっている測定器に接続します

形式：

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

引数：

引数	I/O	説明
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	指定した仮想 COM ポートに接続している測定器との通信が可能になりました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErConnectFailed	エラー	測定器との接続に失敗しました

説明：

Connect()時に comPort に 0 を指定(引数無しも同様)すると、自動で GetDeviceList()を行い最も仮想 COM ポート番号が若い測定器と通信を確立します。以後、各 API で comPort に 0 を指定することで、Connect()した測定器の仮想 COM ポート番号で処理されます。

DisConnect()

概要：

指定した仮想 COM ポートに接続している測定器との通信を切断します

形式：

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

引数：

引数	I/O	説明
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	指定した仮想 COM ポートに接続している測定器との通信を切断しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErDisConnectFailed	エラー	切断に失敗しました

説明：

エラーの場合は測定器の電源をオフにして切断してください。

GetDeviceList()

概要：

PC に接続されている(※１)測定器情報（仮想 COM ポート番号、機種名、シリアルナンバー）のリストを取得します

形式：

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

引数：

引数	I/O	説明
deviceList	O	PC と繋がっている測定器リスト Dictionary の key は仮想 COM ポート番号 value は機種名(シリアルナンバー)

戻り値：

戻り値	種別	説明
KmSuccess	正常	測定器リストを取得できました
ErGetFailed	エラー	測定器リストを取得できませんでした

説明：

例えば COM2 にシリアルナンバーが 12345678 の CS-150 が接続されていれば、deviceList の key に 2,value に”CS-150(12345678)”が格納されます

※１．“PC に接続されている”とは、SDK に接続されているかどうかにかかわらず、PC に接続され、OS に認識されている状態を指します。

4.4 測定

Measure()

概要：

測定を 1 回実行します

形式：

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

引数：

引数	I/O	説明
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	本体が測定を開始しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErMeasurementFailed	エラー	測定に失敗しました

説明：

Enum: MeasStatus

測定状態を表す値定義

MeasStatus	定義値	説明
Idling	0	完了
Measuring	1	測定中

PollingMeasurement()

概要：

測定状態を取得します

形式：

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

引数：

引数	I/O	説明
measStatus	O	測定状態
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	測定状態を取得しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErGetFailed	エラー	測定状態の取得に失敗しました

説明：

CancelMeasurement()

概要：

測定を中断します

形式：

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

引数：

引数	I/O	説明
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	測定を中断しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCancelFailed	エラー	中断に失敗しました

説明：

・測定が実行されていない状態で本関数を実行した場合は、戻り値として「KmSuccess」が戻ってきます。

・CancelMeasurement を実行すると、途中まで実行されていた測定を再開することはできません。

4.5　測定値の取得

ReadLatestData()

概要：

最新の測定による色彩値データを取得します

形式：

ReturnMessage　**ReadLatestData**([任意の表色系クラス] measurementData , Int32 comPort = 0)

引数：

引数	I/O	説明
measurementData	I/O	表色値データ
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	色彩値を取得しました。
ErNoConnect	エラー	指定した COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErNoData	エラー	指定したデータがありません ※1
ErReadFailed	エラー	測定データの取得に失敗しました
ErCalcFailed	エラー	計算に失敗しました ※2
ErOutOfRangeValue	エラー	計算結果の色度もしくは輝度が対応範囲外です ※3

説明：

・取得したい表色系のクラスの変数を引数に指定してください。

・取得できる色彩値データは、最新の測定データに対してのみです。

※1 この場合、計算値は FLT_MAX になります。

※2 この場合、計算値は FLT_MIN になります。

※3 この場合、計算値は対応範囲外の色度もしくは輝度になります。

ReadDisplayValue()

概要：

表示値の取得をします

形式：

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

引数：

引数	I/O	説明
displayValue	O	表示値
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	測定値の測定日時を取得しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErNoData	エラー	取得する値がありません ※1
ErGetFailed	エラー	表示値の取得に失敗しました

説明：

ディスプレイに表示されている値の取得をします。本体の表示値と同じ表色系、データが取得できます。

※　この API で取得する MeasuremetData オブジェクトに含まれる Id と Date には、それぞれ空のデータとデフォルト値が格納されます。

※1 この場合、計算値は FLT_MAX になります。

4.6　測定データ（本体保存）

GetNumberOfSampleData()

概要：

測定器に保存されている測定データの数を取得します

形式：

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

引数：

引数	I/O	説明
dataNum	O	測定データ数
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	測定器に保存されている測定データの数を取得しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErGetFailed	エラー	測定器に保存されている測定データの数の取得に失敗しました

説明：

ReadSampleData()

概要：

測定器に保存されている測定値を取得します。３つのオーバーロード関数（引数の異なる関数）が存在します。

形式１：

ReturnMessage **ReadSampleData**(Int32 dataNumber, [任意の表色系クラス] sample, Int32 comPort = 0)

引数：

引数	I/O	説明
dataNumber	I	サンプルのデータナンバー
sample	I/O	表色値データ
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	測定データを取得しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErNoData	エラー	指定したデータがありません　※1
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErReadFailed	エラー	読み込みに失敗しました
ErCalcFailed	エラー	計算に失敗しました ※2
ErOutOfRangeValue	エラー	計算結果の色度もしくは輝度が対応範囲外です ※3

説明：

取得したい表色系のクラスの変数を引数に指定してください。

※1 この場合、計算値は FLT_MAX になります。

※2 この場合、計算値は FLT_MIN になります。

※3 この場合、計算値は対応範囲外の色度もしくは輝度になります。

形式２：

ReturnMessage **ReadSampleData**(Int32 dataNumber, [任意の表色系クラス] sample, out Int32 targetCh, [任意の表色系クラス] targetData, Int32 comPort = 0)

引数：

引数	I/O	説明
dataNumber	I	サンプルのデータナンバー
sample	I/O	表色値データ
targetCh	O	基準チャンネル
target	I/O	基準値データ
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	測定データを取得しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErNoData	エラー	指定したデータがありません　※1
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErReadFailed	エラー	読み込みに失敗しました
ErCalcFailed	エラー	計算に失敗しました　※2
ErOutOfRangeValue	エラー	計算結果の色度もしくは輝度が対応範囲外です ※3

説明：

測定器に保存されている測定データおよびその値の測定時に基準としていたデータとその基準チャンネルを取得します。

※1 この場合、計算値は FLT_MAX になります。

※2 この場合、計算値は FLT_MIN になります。

※3 この場合、計算値は対応範囲外の色度もしくは輝度になります。

形式３：

ReturnMessage **ReadSampleData**(Int32 dataNumber , [任意の表色系クラス] sample, out Dictionary<string, object> detailedData, Int32 comPort = 0)

引数：

引数	I/O	説明
dataNumber	I	サンプルのデータナンバー
sample	I/O	表色値データ
detailedData	O	基準、任意校正、測定条件データ
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	測定データを取得しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErNoData	エラー	指定したデータがありません　※1
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErReadFailed	エラー	読み込みに失敗しました
ErCalcFailed	エラー	計算に失敗しました　※2
ErOutOfRangeValue	エラー	計算結果の色度もしくは輝度が対応範囲外です　※3

説明：

測定器に保存されている測定データおよび基準値、校正データ、測定条件を取得します。

基準値データや校正データがない場合は detailedData にはそのデータは格納されません。

例：

測定条件

key:“PeakValley”　value:”Peak”

基準値データ

key:"Target"　value: MeasurementData 型のターゲットデータ

校正データ

key:"UserCalibrationData"　value: ColorValueCalibData 型ユーザー校正値

※1 この場合、計算値は FLT_MAX になります。

※2 この場合、計算値は FLT_MIN になります。

※3 この場合、計算値は対応範囲外の色度もしくは輝度になります。

DeleteSampleData()

概要：

測定器に保存されている測定データを削除します

形式：

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

引数：

引数	I/O	説明
dataNumber	I	設定する測定データナンバー -1:全データ削除
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	測定器に保存されている測定データを削除しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErDeleteFailed	エラー	各種設定の削除に失敗しました

説明：

dataNumber に-1 を指定すると、測定器に保存されている全測定データの削除を行います。

※1. データが存在しない dataNumber を指定して削除を実行した場合は、KmSuccess が戻ります。

4.7 基準値

SetTargetCh()

概要：

測定器の基準値チャンネル設定を指定します

形式：

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

引数：

引数	I/O	説明
targetCh	I	基準値チャンネル
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	使用する基準値チャンネルの設定をしました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	指定した基準値チャンネルにデータが存在しないか、無効な値が設定されています
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	使用する基準値チャンネルの設定に失敗しました

説明：

基準値が設定されていない基準値チャンネルは設定できません。

GetTargetCh()

概要：

測定器の基準値チャンネル設定を取得します

形式：

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

引数：

引数	I/O	説明
targetCh	O	基準値チャンネル
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	指定されている基準値チャンネルを取得しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErGetFailed	エラー	指定されている基準値チャンネルの取得に失敗しました

説明：

ReadTargetData()

概要：

測定器に保存されている基準値を取得します

形式：

ReturnMessage **ReadTargetData**(Int32 targetCh, [任意の表色系クラス] target, Int32 comPort = 0)

引数：

引数	I/O	説明
targetCh	I	データを取得したい基準値チャンネル
target	I/O	表色値データ
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	基準値を取得しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErNoData	エラー	指定したチャンネルにはデータがありません ※1
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErReadFailed	エラー	読み込みに失敗しました
ErCalcFailed	エラー	計算に失敗しました ※2
ErOutOfRangeValue	エラー	計算結果の色度もしくは輝度が対応範囲外です ※3

説明：

※1 この場合、計算値は FLT_MAX になります。

※2 この場合、計算値は FLT_MIN になります。

※3 この場合、計算値は対応範囲外の色度もしくは輝度になります。

WriteTargetData()

概要：

測定器に基準値を書き込みます

形式：

ReturnMessage **WriteTargetData**(Int32 targetCh, [任意の表色系クラス] target, Int32 comPort = 0)

引数：

引数	I/O	説明
targetCh	I	基準値チャンネル
target	I	基準値データ
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	基準値を設定しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErWriteFailed	エラー	書き込みに失敗しました
ErOutOfRangeValue	エラー	計算結果の色度もしくは輝度が対応範囲外です ※1

説明：

基準値 ID を入力する場合は target.Id に入力してください。

基準値 ID には以下の半角文字が 12 文字まで使用可能です。

英数字 [0～9] [a～z] [A～Z]

記号 [!"#$%&'()*+,-./:;<=>?@[]¥^_`{|}~[半角スペース]]

※1 この場合、計算値は対応範囲外の色度もしくは輝度になります。

DeleteTargetData()

概要：

測定器に保存されている基準値を削除します

形式：

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

引数：

引数	I/O	説明
targetCh	I	削除する基準値チャンネル -1:全データ削除
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	基準値を削除しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErDeleteFailed	エラー	各種設定の削除に失敗しました

説明：

targetCh に-1 を指定すると、測定器に保存されている全ての基準値を削除します。

※1. データが存在しない targetCh を指定して削除を実行した場合は、KmSuccess が戻ります。

4.8 測定時間・同期設定

Class: MeasurementTime

概要：

測定時間データクラス

形式：

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

変数：

変数	説明
ManualMeasurementTime	Manual 時の測定時間
MeasTimeMode	測定時間モード

説明：

Enum: MeasTimeMode

測定時間モードの値定義

MeasTimeMode	定義値	説明
Auto	0	積算時間自動
Manual	1	積算時間マニュアル

SetMeasurementTime()

概要：

測定器の測定時間設定を指定します

形式：

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

引数：

引数	I/O	説明
measurementTime	I	測定時間と測定時間モード
comport	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	測定時間を設定しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	測定時間の設定に失敗しました

説明：

measurementTime.MeasTimeMode に MeasTimeMode を設定してください。
MeasTimeModeだけ切り替えたい時はmeasurementTime.Timeはデフォルト値のままで設定してください。
MeasTimeMode が Manual(マニュアル測定)の場合は measurementTime.Time に測定時間を設定してください。
measTimeMode が Manual 以外のときは measurementTime.Time は設定に反映されません。

CS-150/CS-160，LS-150/LS-160 接続時、SetSyncMode で SyncMode を Sync（同期測定）に設定した場合は、MeasTimeMode の設定に関わらず、同期周波数に従って測定します

GetMeasurementTime()

概要：

測定器の測定時間設定を取得します

形式

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

引数：

引数	I/O	説明
measurementTime	O	測定時間とモード
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	測定時間を取得しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErGetFailed	エラー	測定時間の取得に失敗しました

説明：

measurementTime.Time には measurementTime.MeasTimeMode に関係なく、
マニュアル測定時の測定時間が格納されます

Class: MeasurementFrequency

概要：

同期/非同期 測定データクラス

形式：

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

変数：

変数	説明
frequency	同期設定時の同期周波数
syncMode	同期/非同期

説明：

Enum: SyncMode

同期モードの値定義

SyncMode	定義値	説明
Async	0	同期測定 OFF
Sync	1	同期測定 ON

SetSyncMode()

概要：

測定器の同期/非同期測定設定を指定します

形式：

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

引数：

引数	I/O	説明
measurementFrequency	I	周波数とモード
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	同期/非同期測定の設定をしました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	同期/非同期測定の設定に失敗しました

説明：

measurementFrequency.SyncMode に SyncMode を設定してください。

SyncMode だけ切り替えたい時は measurementFrequency.Frequency はデフォルト値のままで設定してください。

SyncMode が Sync(同期測定)の場合は measurementFrequency.Frequency に同期時間を設定してください。

measurementFrequency.SyncMode が Async のときは measurementFrequency.Frequency は設定に反映されません。

GetSyncMode()

概要：

測定器の同期/非同期測定設定を取得します

形式：

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

引数：

引数	I/O	説明
measurementFrequency	O	同期周波数と同期測定モード
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	同期/非同期測定の設定を取得しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErGetFailed	エラー	同期/非同期測定の設定に失敗しました

説明：

measurementFrequency.SyncMode に関係なく、measurementFrequency.Frequency には同期測定時の同期周波数が格納されます

4.9 Peak/Valley

Enum: PeakValley

Peaks/Valley 設定を表す値定義

PeakValley	定義値	説明
OFF	0	OFF
Peak	1	Peak 測定
Valley	2	Valley 測定

SetPeakValley()

概要：

測定器の Peak/Valley 設定を指定します

形式：

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

引数：

引数	I/O	説明
peakValley	I	Peak/Valley
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	Peak/Valley の設定をしました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	Peak/Valley の設定に失敗しました

説明：

GetPeakValley()

概要：

測定器の Peak/Valley 設定を取得します

形式：

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

引数：

引数	I/O	説明
peakValley	O	Peak/Valley
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	Peak/Valley の設定を取得しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErGetFailed	エラー	Peak/Valley の設定の取得に失敗しました

説明：

4.10 レンズ設定

Enum: CloseUpLensType

クローズアップレンズ設定の値定義

CloseUpLensType	定義値	説明
Standard	0	標準(なし)
No153	1	No.153
No135	2	No.135
No122	3	No.122
No110	4	No.110

SetCloseUpLens()

概要：

測定器のクローズアップレンズ設定を指定します

形式：

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

引数：

引数	I/O	説明
closeUpLensType	I	クローズアップレンズの種類
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	クローズアップレンズの設定をしました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	クローズアップレンズの設定に失敗しました

説明：

GetCloseUpLens()

概要：

測定器のクローズアップレンズ設定を取得します

形式：

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

引数：

引数	I/O	説明
closeUpLensType	O	クローズアップレンズ設定
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	クローズアップレンズの設定を取得しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErGetFailed	エラー	クローズアップレンズの設定に失敗しました

説明：

4.11　ユーザー校正チャンネル

SetCalibrationCh()

概要：

測定器のユーザー校正チャンネルを指定します

形式：

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

引数：

引数	I/O	説明
calibrationCh	I	ユーザー校正チャンネル
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	使用するユーザー校正チャンネルを設定しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	指定した校正データチャンネルにデータが存在しないか、無効な値が設定されています
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	使用するユーザー校正チャンネルの設定に失敗しました

説明：

ユーザー校正値が設定されていないユーザー校正チャンネルは設定できません。

GetCalibrationCh()

概要：

測定器に設定されているユーザー校正チャンネルを取得します

形式：

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

引数：

引数	I/O	説明
calibrationCh	O	ユーザー校正チャンネル
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	指定されているユーザー校正チャンネルを取得しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErGetFailed	エラー	指定されているユーザー校正チャンネルの取得に失敗しました

説明：

4.12 ユーザー校正

Class: UserCalibData

概要：

ユーザー校正に用いるデータクラス

形式：

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

変数：

変数	説明
Id	ID
Date	登録日時
CalibType	校正種別
Coef	校正係数

説明：

Enum: CalibType

校正タイプを表す値定義

CalibType	定義値	説明
OnePoint	0	1 点校正
RGB	1	RGB 校正
WRGB	2	WRGB 校正

SetMatrixCalib()

概要：

測定器にユーザー校正係数を設定します

形式：

ReturnMessage **SetMatrixCalib**(Int32 calibrationCh, List<[任意の表色系クラス]> measurementDataList, List<[任意の表色系クラス]> correctDataList, CalibType type,string id, Int32 comPort = 0)

引数：

引数	I/O	説明
calibrationCh	I	ユーザー校正チャンネル
measurementDataList	I	測定値(校正前値)リスト
correctDataList	I	真値(校正後値)リスト
type	I	校正種別 （1 点校正、RGB 校正、WRGB 校正）
id	I	ID
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	ユーザー校正係数を設定しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	ユーザー校正係数の設定に失敗しました
ErOutOfRangeValue	エラー	計算結果の色度もしくは輝度が対応範囲外です ※1

説明：

【1 点校正】

type = CalibType.One Point
measurementDataList[0] = W を測定した表色値
correctDataList は measurementDataList と同様に表色値を設定してください。
※ ゼロに近い小さな値を校正値として用いた場合など、1 点校正の計算が不可能な係数には 1 が設定されます。

【RGB 校正の場合】
type = CalibType.RGB
measurementDataList[0] = R を測定した表色値
measurementDataList[1] = G を測定した表色値
measurementDataList[2] = B を測定した表色値
correctDataList は measurementDataList と同様に表色値を設定してください。

【WRGB 校正の場合】
type = CalibType.WRGB
measurementDataList[0] = W を測定した表色値
measurementDataList[1] = R を測定した表色値
measurementDataList[2] = G を測定した表色値
measurementDataList[3] = B を測定した表色値
correctDataList は measurementDataList と同様に表色値を設定してください。
※ RGB 校正と WRGB 校正を実行する際に、逆行列が存在しない場合があります。この場合は、errorCode として ErSetFailed が返り、errorMessage には"Inverse matrix does not exist."が格納されます。

また、校正日時は自動で PC の時刻を書き込みます。

ID には以下の半角文字が 12 文字まで使用可能です。
英数字 [0～9] [a～z] [A～Z]
記号 [!"#$%&'()*+,-./:;<=>?@[]¥^_`{|}~[半角スペース]]

※1 この場合、計算値は対応範囲外の色度もしくは輝度になります。

GetCalibData()

概要：

測定器のユーザー校正チャンネルに設定されている校正パラメータを取得します

形式：

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

引数：

引数	I/O	説明
calibrationCh	I	ユーザー校正チャンネル
measurementDataList	O	ユーザー校正係数設定時に使用された測定値
correctDataList	O	ユーザー校正係数設定時に使用された真値
calibData	O	取得するユーザー校正データ
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	ユーザー校正係数を取得しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErNoData	エラー	指定した ch にはデータがありません ※1
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErGetFailed	エラー	ユーザー校正係数を取得に失敗しました
ErCalcFailed	エラー	計算に失敗しました ※2
ErOutOfRangeValue	エラー	計算結果の色度もしくは輝度が対応範囲外です ※3

説明：

・取得した測定値・真値の輝度単位は、測定器の設定と同じになります。

※1 この場合、計算値は FLT_MAX になります。

※2 この場合、計算値は FLT_MIN になります。

※3 この場合、計算値は対応範囲外の色度もしくは輝度になります。

DeleteCalibData()

概要：

測定器に保存されているユーザー校正係数を削除します

形式：

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

引数：

引数	I/O	説明
calibrationCh	I	削除する校正 ch -1:全データ削除
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	ユーザー校正係数を削除しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErDeleteFailed	エラー	ユーザー校正係数の削除に失敗しました

説明：

calibrationCh に-1 を指定すると、測定器に保存されている全ユーザー校正係数を削除します

※1. データが存在しない calibrationCh を指定して削除を実行した場合は、KmSuccess が戻ります。

4.13 CCF (Color Correction Factor)

被測定光源等の色補正係数があらかじめ分かっている場合は、その補正係数を本体に設定して補正された測定を実行させることができます。

Class: ColorCorrectionFactor

概要：

色補正係数データクラス

形式：

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

変数：

変数	説明
Coef	色補正係数値
CcfMode	オン/オフ

説明：

Enum: CCFMode

CCFMode	定義値	説明
OFF	0	OFF
ON	1	ON

SetCCF()

概要：

CCF（色補正係数）設定を指定します

形式：

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

引数：

引数	I/O	説明
ccfData	I	色補正係数値と On/Off
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	色補正係数と On/ff を設定しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	色補正係数と On/Off の設定に失敗しました

説明：

色補正係数の On/Off だけ切り替えたい時は ccfData.Ccf はデフォルト値のままで設定してください。

GetCCF()

概要：

CCF（色補正係数）設定を取得します

形式：

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

引数：

引数	I/O	説明
ccfData	O	色補正係数値と On/Off
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	色補正係数と On/Off を取得しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErGetFailed	エラー	色補正係数と On/Off の取得に失敗しました

説明：

4.14　電源設定

Enum: AutoPowerOff

オートパワーオフ設定の値定義

AutoPowerOff	定義値	説明
Off	0	オートパワーオフ Off
On	1	オートパワーオフ On

SetAutoPowerOff()

概要：

測定器のオートパワーオフ設定を指定します

形式：

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

引数：

引数	I/O	説明
autoPowerOff	I	オートパワーOff 設定
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	オートパワーOff の設定をしました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	オートパワーOff の設定に失敗しました

説明：

GetAutoPowerOff()

概要：

測定器のオートパワーオフ設定を取得します

形式：

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

引数：

引数	I/O	説明
autoPowerOff	O	オートパワーOff 設定
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	オートパワーOff の設定をしました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErGetFailed	エラー	オートパワーOff の設定の取得に失敗しました

説明：

4.15 バックライト

Enum: BackLightMode

バックライトの設定の値定義

BackLightMode	定義値	説明
Off	0	バックライトオフ
On	1	バックライトオン

SetBackLightOnOff()

概要：

測定器のバックライト On/Off 設定を指定します

形式：

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

引数：

引数	I/O	説明
backLightMode	I	バックライト On/Off
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	バックライト On/Off の設定をしました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErSetFailed	エラー	各種設定に失敗しました

説明：

GetBackLightOnOff()

概要：

測定器のバックライト On/Off 設定を指定します

形式：

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

引数：

引数	I/O	説明
backLightMode	O	バックライト On/Off
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	バックライト On/Off の設定を取得しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErGetFailed	エラー	バックライト On/Off の設定の取得に失敗しました

説明：

Enum: BackLightLevel

バックライトの設定の値定義

BackLightLevel	定義値	説明	
Level1	1	明るさ１	暗い ↕ 明るい
Level2	2	明るさ２	
Level3	3	明るさ３	
Level4	4	明るさ４	
Level5	5	明るさ５	

SetBackLightLevel()

概要：

測定器のバックライトの明るさ設定を指定します

形式：

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

引数：

引数	I/O	説明
backLightLevel	I	バックライトの明るさ
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	バックライトの明るさを設定しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	バックライトの明るさの設定に失敗しました

説明：

GetBackLightLevel()

概要：

測定器のバックライトの明るさ設定を取得します

形式：

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

引数：

引数	I/O	説明
backLightLevel	O	バックライト設定
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	バックライトの明るさ設定を取得しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErGetFailed	エラー	バックライトの明るさ設定の取得に失敗しました

説明：

4.16　表示桁数

SetColorDispDigit()

概要：

測定器の色度表示桁数設定を指定します

形式：

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

引数：

引数	I/O	説明
digitNum	I	色度表示桁数
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	色度表示桁数の設定をしました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	色度表示桁数の設定に失敗しました

説明：

3 桁を設定したいときは digitNum = 3;

4 桁を設定したいときは digitNum = 4;

を格納してください。

GetColorDispDigit()

概要：

測定器の色度表示桁数設定を取得します

形式：

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

引数：

引数	I/O	説明
digitNum	O	色度表示桁数
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	色度表示桁数の設定を取得しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErSetFailed	エラー	色度表示桁数の設定に失敗しました

説明：

4.17 表示形式（絶対値、差分、比率）

Enum: DispType

測定器画面の表示形式設定を表す値定義

DispType	定義値	説明
Abs	0	絶対値表示
Diff	1	差分表示
Ratio	2	比率表示

SetDisplayType()

概要：

本体ディスプレイの表示形式設定(絶対値、差分、比率)を指定します

形式：

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

引数：

引数	I/O	説明
dispType	I	ディスプレイ表示形式
comport	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	表示形式を設定しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	各種設定に失敗しました

説明：

GetDisplayType()

概要：

本体ディスプレイの表示形式設定(絶対値、差分、比率)を取得します

形式：

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

引数：

引数	I/O	説明
dispType	O	ディスプレイ表示形式
comport	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	表示形式を取得しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErGetFailed	エラー	表示形式の取得に失敗しました

説明：

4.18　言語・日時

Enum: DisplayLanguage

Set/GetLanguage()で言語の設定・取得の際に利用する値定義です

DisplayLanguage	定義値	説明
ENG	0	英語(米)
JPN	1	日本語
CHI	2	中国語(簡体)

SetDisplayLanguage()

概要：

測定器の言語設定を指定します

形式：

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

引数：

引数	I/O	説明
displayLanguage	I	言語
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	指定された言語を設定しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	言語設定に失敗しました

説明：

GetDisplayLanguage()

概要：

測定器の言語設定を取得します

形式：

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

引数：

引数	I/O	説明
displayLanguage	O	言語
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	言語設定を取得しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErGetFailed	エラー	言語設定の取得に失敗しました

説明：

Class: DateTime

概要：

日付・時刻クラス

System 名前空間で提供されるクラスです。

形式：

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

説明：

日付・時刻の設定取得に使用します

SetDateTime()

概要：

測定器の日時設定を指定します

形式：

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

引数：

引数	I/O	説明
dateTime	I	日時
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	日時の設定をしました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	日時の設定に失敗しました

説明：

dateTime には年月日時分秒の６項目を設定してください。

GetDateTime()

概要：

測定器の日時設定を取得します

形式：

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

引数：

引数	I/O	説明
dataTime	O	日時
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	成功
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErGetFailed	エラー	各種設定の取得に失敗しました

説明：

Enum: DateFormat

日付フォーマットの値定義

DateFormat	定義値	説明
YYMMDD	0	年/月/日
MMDDYY	1	月/日/年
DDMMYY	2	日/月/年

SetDateFormat()

概要：

測定器の日付フォーマット設定を指定します

形式：

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

引数：

引数	I/O	説明
dateFormat	I	日付フォーマット
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	日付フォーマットを設定しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	日付フォーマットの設定に失敗しました

説明：

GetDateFormat()

概要：

測定器の日付フォーマット設定を取得します

形式：

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

引数：

引数	I/O	説明
dateFormat	O	日付フォーマット
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	日付フォーマットを取得
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErGetFailed	エラー	日付フォーマットの取得に失敗しました

説明：

4.19 表色モード本体表示

Enum: ColorMode

測定器に表示される表色モードを表す値定義

ColorMode	定義値	説明
Lvxy	0	Lv、x、y
Lvudvd	1	Lv、u′、v′
LvTcpDuv	2	Lv、相関色温度 Tcp、duv
XYZ	3	X、Y、Z
LvDwPe	4	Lv、主波長λd、刺激純度 Pe
Lv	5	Lv

SetColorMode()

概要：

表色モードを設定を指定します

形式：

ReturnMessage　SetColorMode(ColorMode colorMode, Int32 comPort = 0)

引数：

引数	I/O	説明
colorMode	I	表色モード
comport	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	表色モードを設定しました。
ErNoConnect	エラー	指定した指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	引数 colorMode で指定した表色モードは、無効な値が指定されているか、本体側の設定で Off になっています
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	各種設定に失敗しました

説明：

本体ディスプレイに表示する表色モードを設定します。

引数 ColorMode で指定した表色モードは、あらかじめ SetColorModeDisplayOnOff()で表示 ON にしておく必要があります。

GetColorMode()

概要：

表色モード設定を取得します

形式：

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

引数：

引数	I/O	説明
colorMode	O	表色モード
comport	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	表色モードを取得しました。
ErNoConnect	エラー	指定した指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErGetFailed	エラー	各種設定に失敗しました

説明：

本体ディスプレイに表示する表色モードを取得します。

Enum: ColorModeDisplay

ColorModeDisplay	定義値	説明
Off	0	OFF
On	1	ON

SetColorModeDisplayOnOff()

概要：

本体ディスプレイに表示する、任意の表色モードの On/Off を設定します

形式：

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

引数：

引数	I/O	説明
colorMode	I	表色モード
onOff	I	OnOff
comport	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	任意の表色モードの On/Off を設定しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	任意の表色モードの On/Off を設定に失敗しました

説明：

引数の colorMode には、接続している機種が扱うことができる表色モードのみ設定できます。

GetColorModeDisplayOnOff()

概要：

本体ディスプレイに表示する、任意の表色モードの On/Off の設定を取得します

形式：

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

引数：

引数	I/O	説明
colorMode	I	表色モード
onOff	O	OnOff
comport	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	任意の表色モードの On/Off の設定を取得しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErGetFailed	エラー	任意の表色モードの On/Off の設定の取得に失敗しました

説明：

4.20 保存設定

Enum: DataSaveMode

DataSaveMode	定義値	説明
AutoSave	0	自動
ManuSave	1	マニュアル

SetDataSaveMode()

概要：

測定器のデータ保存方法設定を指定します

形式：

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

引数：

引数	I/O	説明
dataSaveMode	I	データ保存方法
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	データ保存方法の設定をしました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	データ保存方法の設定に失敗しました

説明：

GetDataSaveMode()

概要：

測定器のデータ保存方法設定を取得します

形式：

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

引数：

引数	I/O	説明
dataSaveMode	O	データ保存方法
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	データ保存方法の設定を取得しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErGetFailed	エラー	データ保存方法の設定を取得に失敗しました

説明：

4.21 定期校正設定

Enum: PeriodicCalibNotify

Set/GetPeriodicCalibNotify()で定期校正警告設定の設定・取得する際に利用する値定義です

PeriodicCalibNotify	定義値	説明
On	0	定期校正警告 ON
Off	1	定期校正警告 OFF

SetPeriodicCalibNotify()

概要：

測定器の定期校正警告 On/Off 設定を指定します

形式：

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

引数：

引数	I/O	説明
periodicCalibNotify	I	定期校正警告設定
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	定期校正警告 On/Off の設定をしました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	定期校正警告 On/Off の設定に失敗しました

説明：

GetPeriodicCalibNotify()

概要：

測定器の定期校正警告 On/Off 設定を取得します

形式：

ReturnMessage GetPeriodicCalibNotify(out PeriodicCalibNotify periodicCalibNotify, Int32 comPort)

引数：

引数	I/O	説明
periodicCalibNotify	O	定期校正警告設定
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	定期校正警告 On/Off の設定を取得しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErGetFailed	エラー	定期校正警告 On/Off の設定の取得に失敗しました

説明：

4.22 ボタン設定

Enum: ToggleStatus

トグル設定の値定義

ToggleStatus	定義値	説明
Off	0	測定ボタントグル OFF(標準)
On	1	測定ボタントグル ON(トグル方式)

SetToggleOnOff()

概要：

測定器のトグル On/Off 設定を指定します

形式：

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

引数：

引数	I/O	説明
toggleStatus	I	トグル設定
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	測定ボタンのトグル On/Off を設定しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	測定ボタンのトグル On/Off 設定に失敗しました

説明：

ToggleStatus がオンの時、
測定ボタンを押すと測定が始まり、もう一度測定ボタンを押すと測定が止まります。

ToggleStatus がオフの時、
測定ボタンを押すと測定が始まり、そのまま測定ボタンを押し続けていると測定され続け、
測定ボタンを離すと測定が止まります。

GetToggleOnOff()

概要：

測定器のトグル On/Off 設定を取得します

形式：

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

引数：

引数	I/O	説明
toggleStatus	O	トグル設定
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	測定ボタンのトグル On/Off 設定を取得しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErGetFailed	エラー	測定ボタンのトグル On/Off 設定の取得失敗しました

説明：

Enum: TriggerStatus

ボタン状態の値定義

TriggerStatus	定義値	説明
Off	0	測定ボタン操作無効
On	1	測定ボタン操作有効

SetTriggerOnOff()

概要：

測定器の測定ボタン有効無効設定を指定します

形式：

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

引数：

引数	I/O	説明
triggerStatus	I	測定ボタンの有効無効
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	測定ボタンの有効無効の設定を取得しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErSetFailed	エラー	測定ボタンの有効無効の設定に失敗しました

説明：

TriggerStatus がオフの場合、測定器のトリガ(測定ボタン)を使用した測定ができません。

測定器を切断すると、測定ボタンは自動的に無効になります。トリガを使用した測定をするためには、接続のたびに本 API を実行する必要があります。

GetTriggerOnOff()

概要：

測定器の測定ボタン有効無効設定を指定します

形式：

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

引数：

引数	I/O	説明
triggerStatus	O	測定ボタンの有効無効
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	測定ボタンの有効無効の設定を取得しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErGetFailed	エラー	各種設定の取得に失敗しました

説明：

測定ボタンの有効無効の設定を取得します

4.23 本体・SDK 情報取得

Class: DeviceInfo

概要：

測定器本体情報クラス

形式：

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

変数：

変数	説明
ProductName	製品名称
SerialNumber	本体のシリアル No.を格納します
SoftMajorVersion	ファームウェアのメジャーバージョン情報を格納します
SoftMinorVersion	ファームウェアのマイナーバージョン情報を格納します
SoftFreeVersion	ファームウェアのフリーバージョン情報を格納します
PeriodicCalibrationExpirationDate	定期校正満了日を格納します
PeriodicCalibrationWarningStatus	定期校正警告状態を ture,false で格納します true の場合は定期校正をお勧めします。

説明：

GetDeviceInfo()で本体情報を取得する際に利用します。

GetDeviceInfo()

概要：

製品名称や定期校正満了日等の測定器情報を取得します

形式：

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

引数：

引数	I/O	説明
deviceInfo	O	測定器情報
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	測定器情報を取得しました。
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErInvalidParameter	エラー	設定値が不正な値です
ErInstrumentProcessing	エラー	測定器が処理中なのでコマンドを受け付けません
ErGetFailed	エラー	測定器情報の取得に失敗しました

説明：

GetSDKVersion()

概要：

SDK のバージョン情報を取得します

形式：

ReturnMessage **GetSDKVersion**(out string SDKFileVersion)

引数：

引数	I/O	説明
SDKFile	O	O:SDK バージョン

戻り値：

戻り値	種別	説明
KmSuccess	正常	バージョン情報を取得しました。
ErGetFailed	エラー	各種設定の取得に失敗しました

説明：

“LC-MISDK：1.0.0.0”のように出力されます

4.24 単位設定

Enum: LuminanceUnit

輝度単位を表す定義値

LuminanceUnit	定義値	説明
cdm2	0	cd/m^2
other	1	

GetLuminanceUnit()

概要：

測定器の輝度単位設定を指定します

形式：

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

引数：

引数	I/O	説明
luminanceUnit	O	輝度単位
comPort	I	仮想 COM ポート番号

戻り値：

戻り値	種別	説明
KmSuccess	正常	輝度単位の設定を取得しました
ErNoConnect	エラー	指定した仮想 COM ポートに測定器が接続されていません
ErCannotCommand	エラー	指定されたコマンドはこの機種では対応していません
ErGetFailed	エラー	輝度単位の設定の取得に失敗しました

説明：

5. Appendix

5.1 エラーコード一覧

ErrorDefine	定義値	原因
KmSuccess	0	処理が正常に完了しました。
ErNoConnect	10	指定した仮想 COM ポート番号に接続している測定器がありません
ErInvalidParameter	25	代入したパラメータが正しくありません
ErCannotCommand	30	指定されたコマンドはこの機種では対応していません
ErNoData	45	指定したデータがありません
ErOutOfRangeValue	50	計算結果の色度もしくは輝度が対応範囲外です
ErInstrumentProcessing	60	現在測定器が処理中なのでコマンドを受け付けません
ErConnectFailed	100	測定器との接続に失敗しました
ErDisConnectFailed	110	１つ以上切断できないポートがありました
ErSetFailed	120	各種設定に失敗しました
ErGetFailed	130	各種設定の取得に失敗しました
ErCalcFailed	140	計算に失敗しました
ErCancelFailed	150	中断に失敗しました
ErWriteFailed	160	書き込みに失敗しました
ErReadFailed	170	読み込みに失敗しました
ErDeleteFailed	180	各種設定の削除に失敗しました
ErMeasurementFailed	200	測定に失敗しました

5.2 表色系クラス

扱える表色系は機種によって異なります。

Class: MeasurementData

概要：

各表色系クラスの基底クラス

形式：

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

変数：

変数	説明
Id	ID
ColorModeId	表色系 ID
Date	登録・測定日時
ColorSpaceValue	表色値

説明：

Class: XYZ

概要：

（ X *Y Z* ）クラス

形式：

```
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; }
        }
}
```

変数：

変数	説明
X	三刺激値 X
Y	三刺激値 Y
Z	三刺激値 Z

説明：

表色値を一つずつ設定取得

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

表色値を foreach で取得

```
foreach(var colorValue in xyz.ColorSpaceValue){
```

```
    WriteLine(colorValue.Value);
}
```

Class: Lvxy

概要：

（ L_v x y ）クラス

形式：

```
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;

}
```

変数：

変数	説明
Lv	輝度
x	色度 x
y	色度 y
unit	輝度単位

説明：

Lvxy lvxy = new Lvxy();

または

Lvxy lvxy = new Lvxy(LuminanceUnit.cdm2);

Class: Lvudvd

概要：

（ L_v u' v' ）クラス

形式：

```
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;
    }
```

変数：

変数	説明
Lv	輝度
ud	CIE 1976 UCS 色度図座標 u'
vd	CIE 1976 UCS 色度図座標 v'
unit	輝度単位

説明：

Lvudvd lvudvd = new Lvudvd();

または

Lvudvd lvudvd = new Lvudvd(LuminanceUnit.cdm2);

Class: LvTcpDuv

概要：

（輝度、相関色温度 T、黒体軌跡からの色差 duv）クラス

形式：

```
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;
}
```

変数：

変数	説明
Lv	輝度
Tcp	相関色温度 T
Duv	黒体軌跡からの色差 duv
unit	輝度単位

説明：

LvTcpDuv lvTcpDuv = new LvTcpDuv();
または
LvTcpDuv lvTcpDuv = new LvTcpDuv(LuminanceUnit.cdm2);

※1. WriteTargetData や SetMatrixCalib などの API を利用して色温度のデータを本体に書き込んだ場合、一部の機種(CS150 シリーズなど)では計算誤差が発生します。これは、一部の機種ではデータを XYZ 形式で管理しているため、書き込み時の変換演算誤差が避けられないために起こります。
誤差量を抑えるため、-0.02 < duv < 0.02 の範囲で使用することをお勧めします。

Class: LvDwPe

概要：

（輝度、主波長 λ_d、刺激純度）クラス

形式：

```
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;
}
```

変数：

変数	説明
Lv	輝度
Dw	主波長 λ_d [nm]
Pe	刺激純度 [%]
unit	輝度単位

説明：

LvDwPe lvDwPe = new LvDwPe();

または

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

※1　主波長(Dw)がマイナスの場合は補色主波長を表します。

Class: Lv

概要：

輝度クラス

形式：

```
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;
}
```

変数：

変数	説明
Lv	輝度
unit	輝度単位

説明：

Lv lv = new Lv();

または

Lv lv = new Lv(LuminanceUnit.cdm2);

5.3 デバイスドライバーのインストール

測定器を USB 接続するには、あらかじめデバイスドライバーをインストールする必要があります。

初めに、PC に測定器を接続し、測定器の電源を ON にすると、ドライバーのインストールが開始されます。

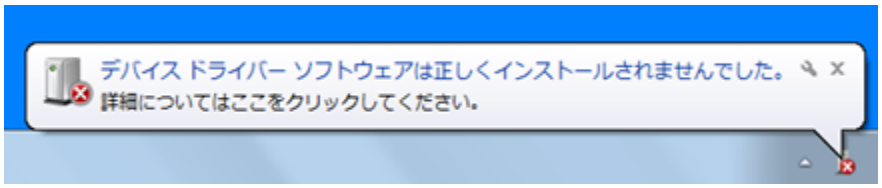
また、Windows10 によってインストールされるデバイスドライバーは正しく動作しない可能性があります。このため、下記のインストール手順に従ってデバイスドライバファイル"KMMIUSB.INF"を手動でインストールしてください。

自動インストール

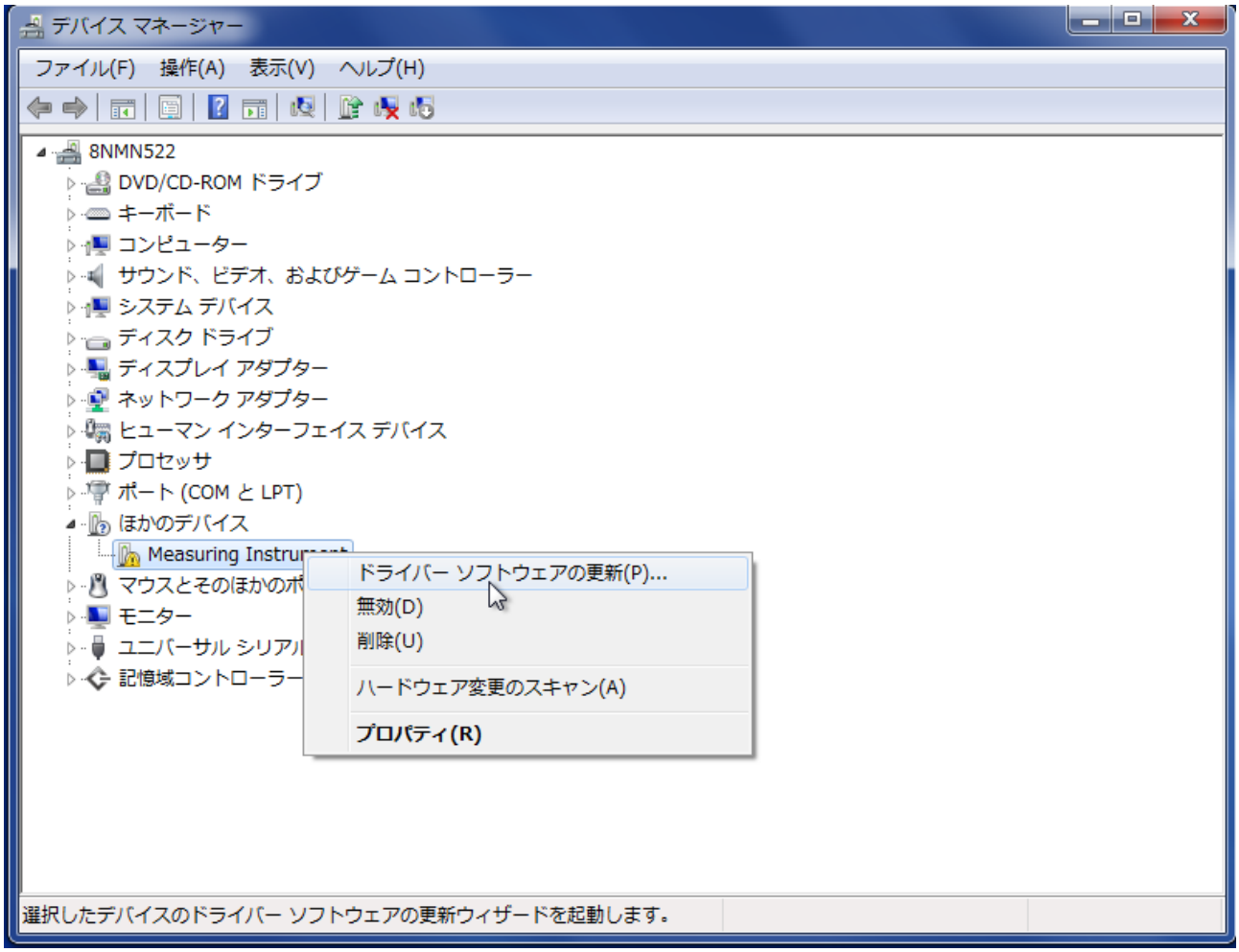
自動インストールに成功した場合は、インストール終了です。

手動インストール

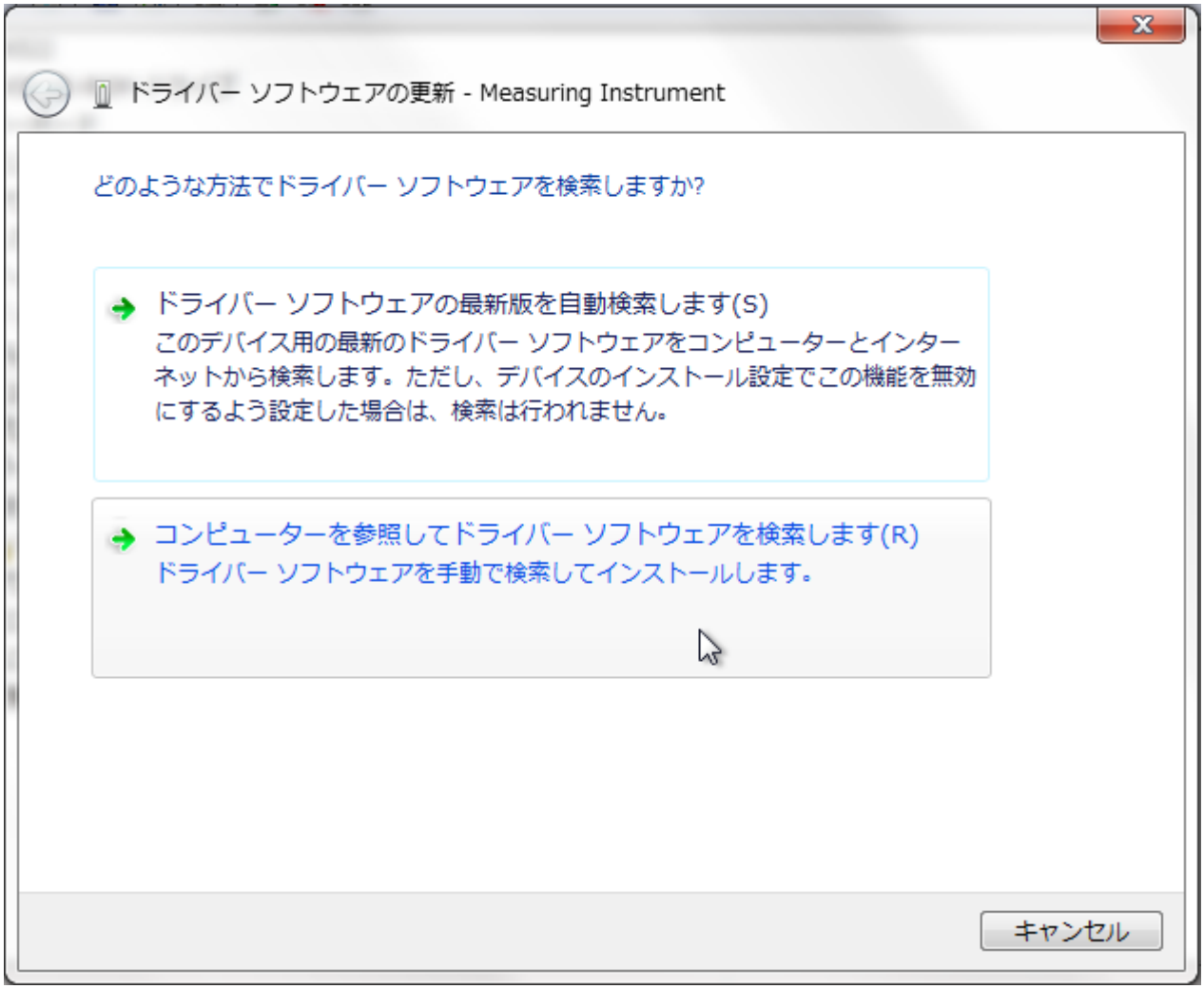
自動インストールに失敗した場合は、以下の手順で手動インストールを実行してください。



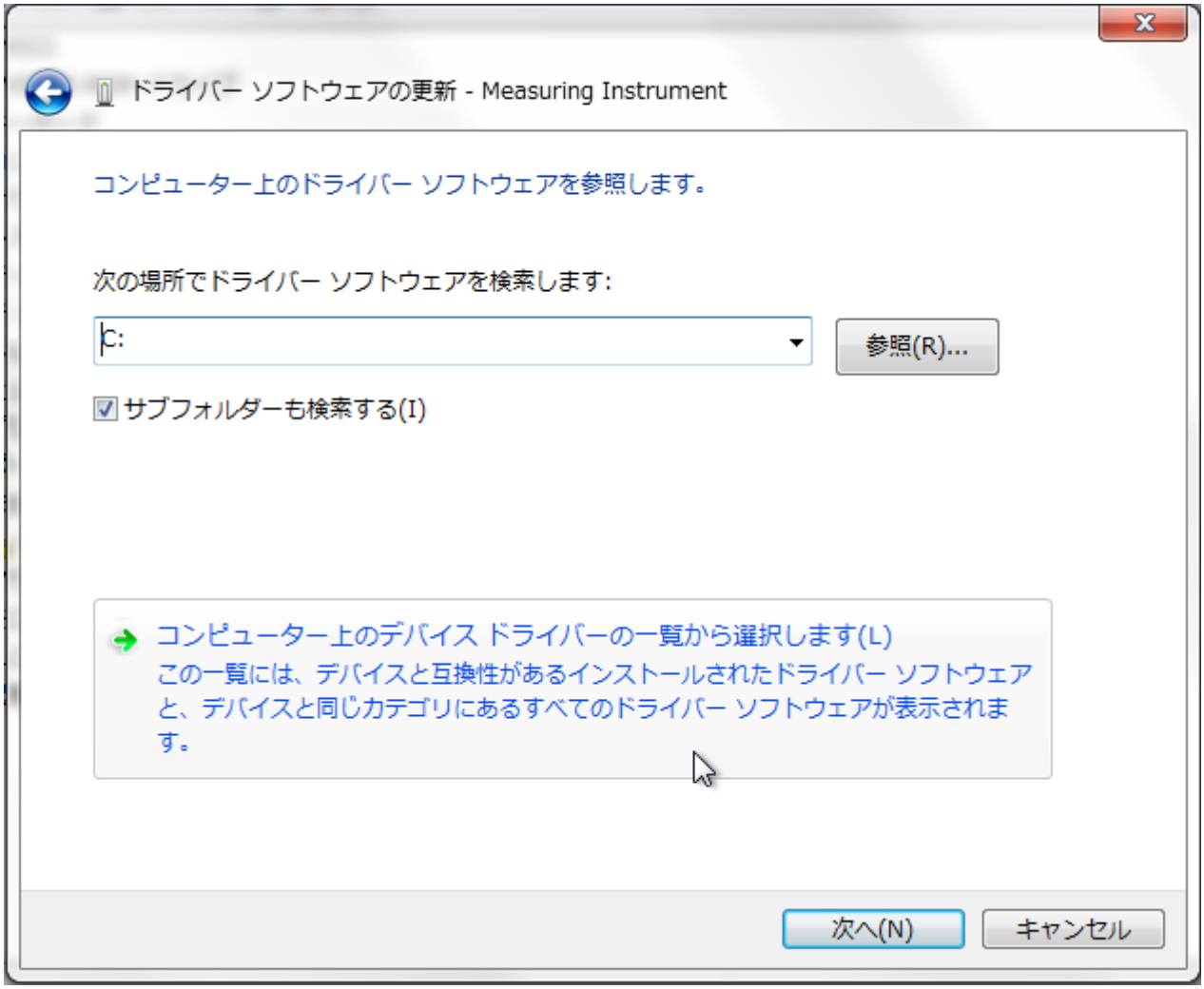
デバイスマネージャーを開き、[ほかのデバイス]-[Measuring Instruments]を右クリックし、ドライバーソフトウェアの更新をクリックします。ドライバーインストールが失敗している状態では、下図のように[Measuring Instruments]に警告マークが付いています。



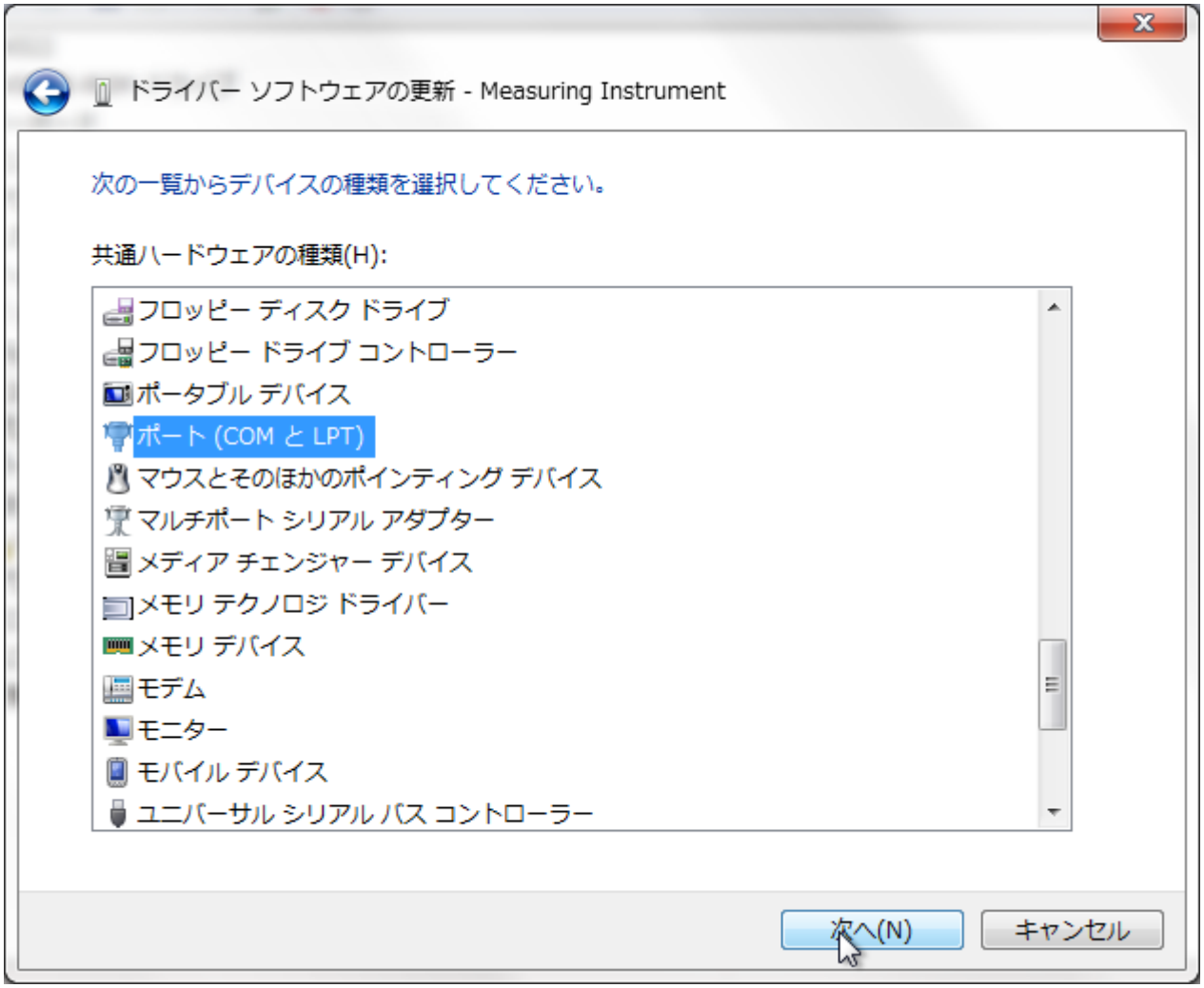
「コンピューターを参照してドライバーソフトウェアを検索します」をクリックします。



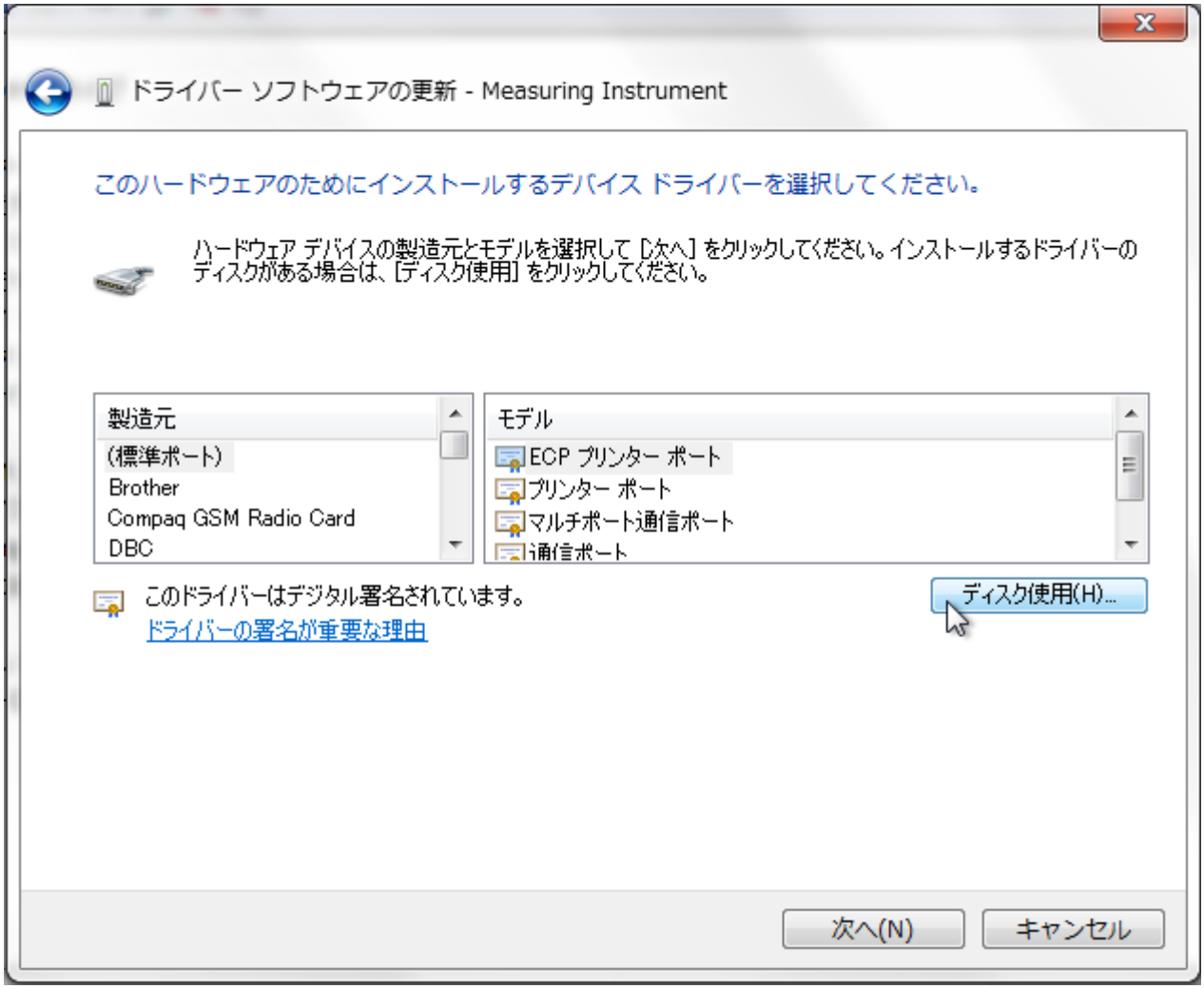
「コンピューター上のデバイスドライバーの一覧から選択します」をクリックします。



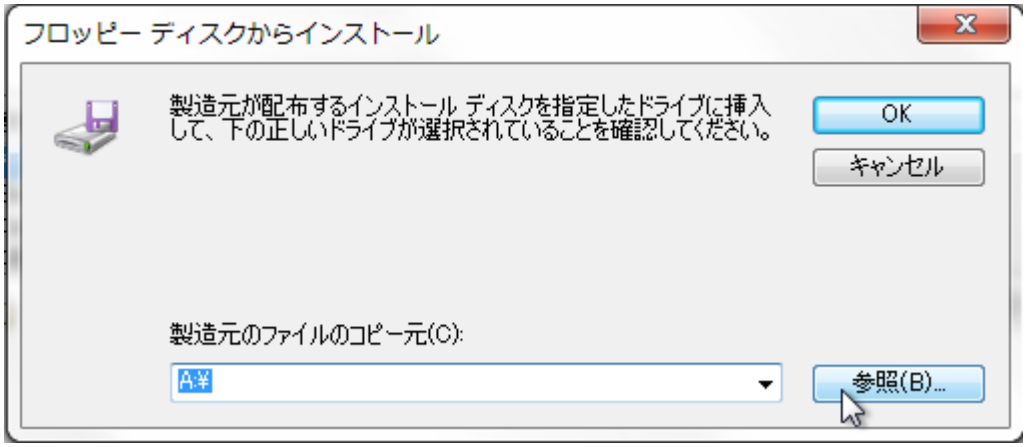
共通ハードウェアの種類一覧から、「ポート(COM と LPT)」をクリックし、「次へ」をクリックします。



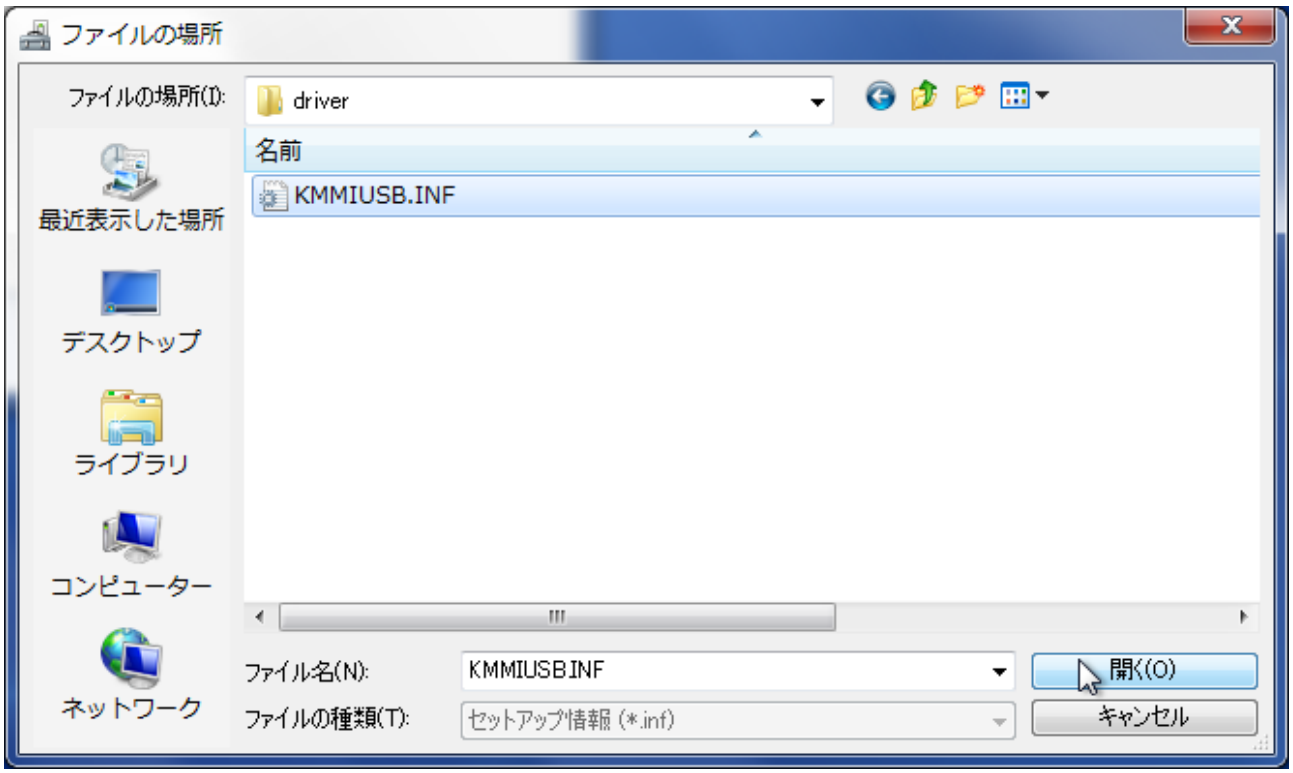
ディスク使用をクリックします。



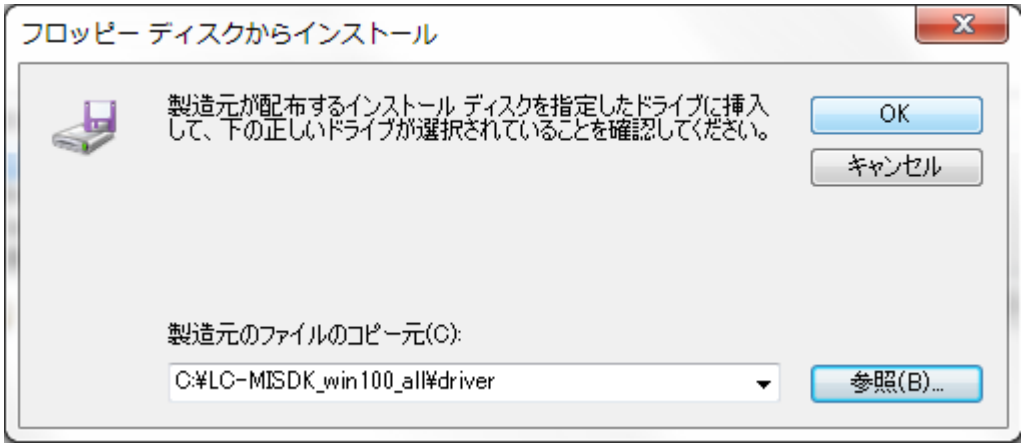
参照をクリックし、



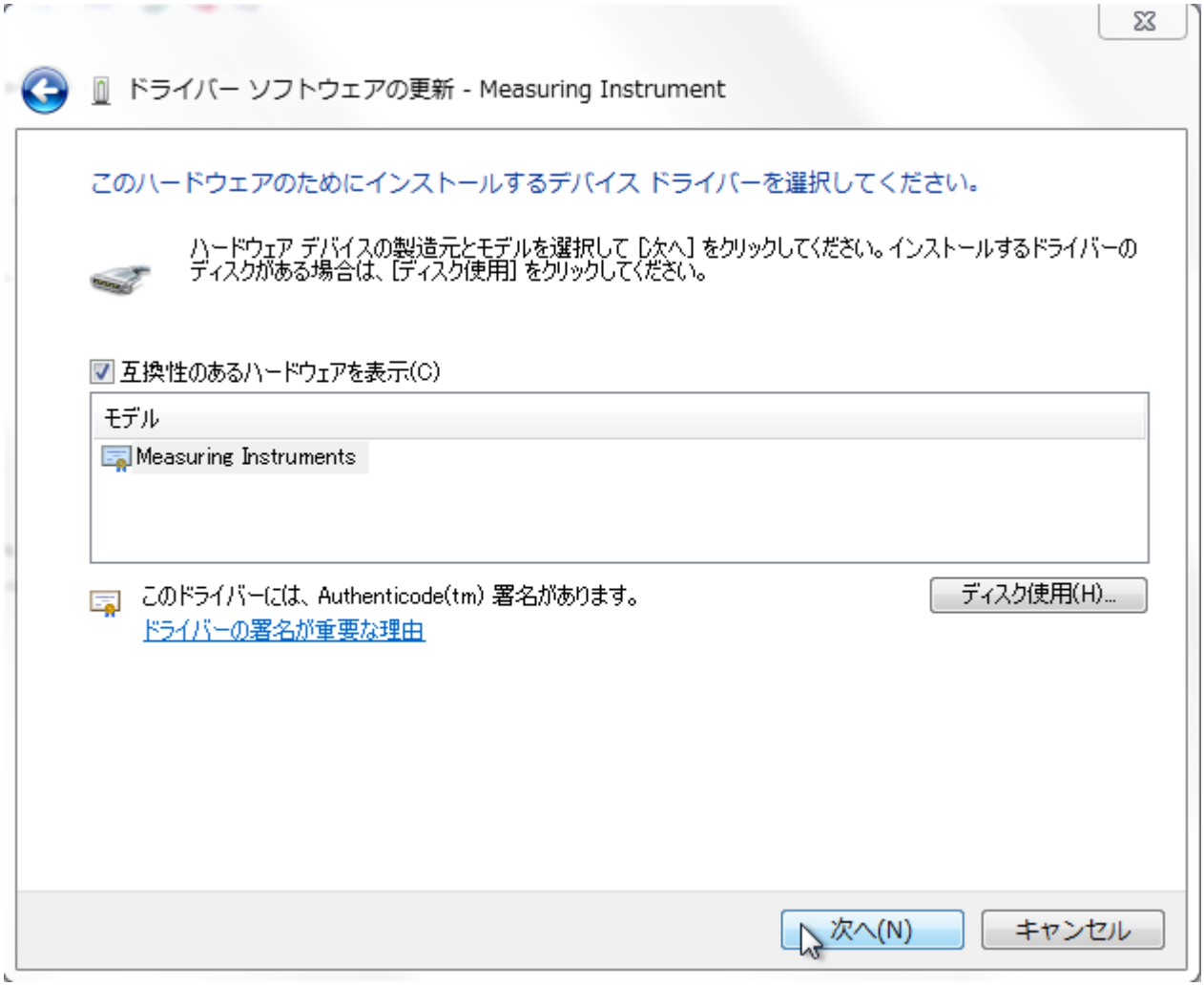
LC-MISDK_win100_all/driver/にある「KMMIUSB.INF」ファイルを選択し、「開く」をクリックします。



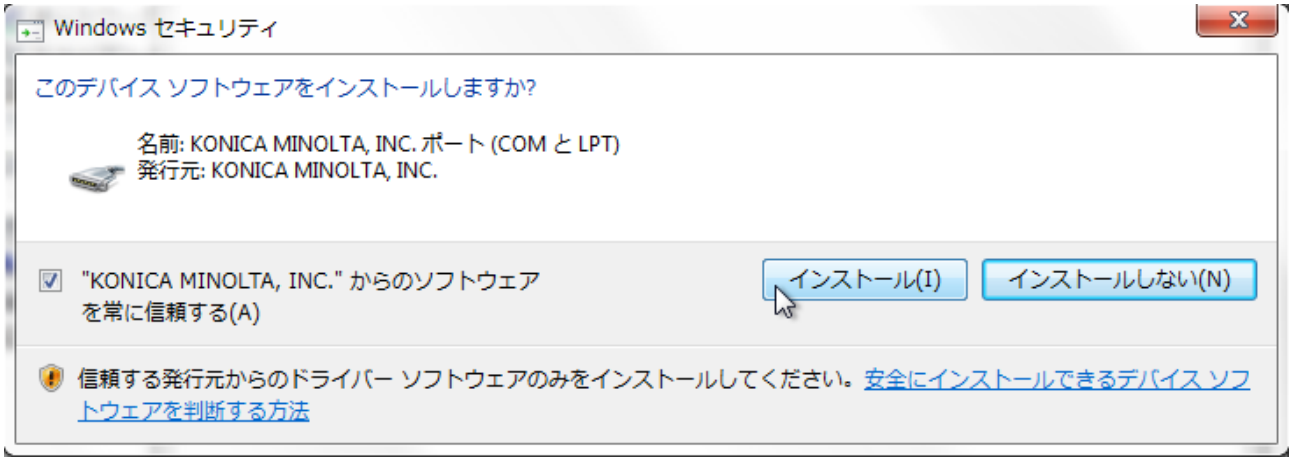
「OK」をクリックします。



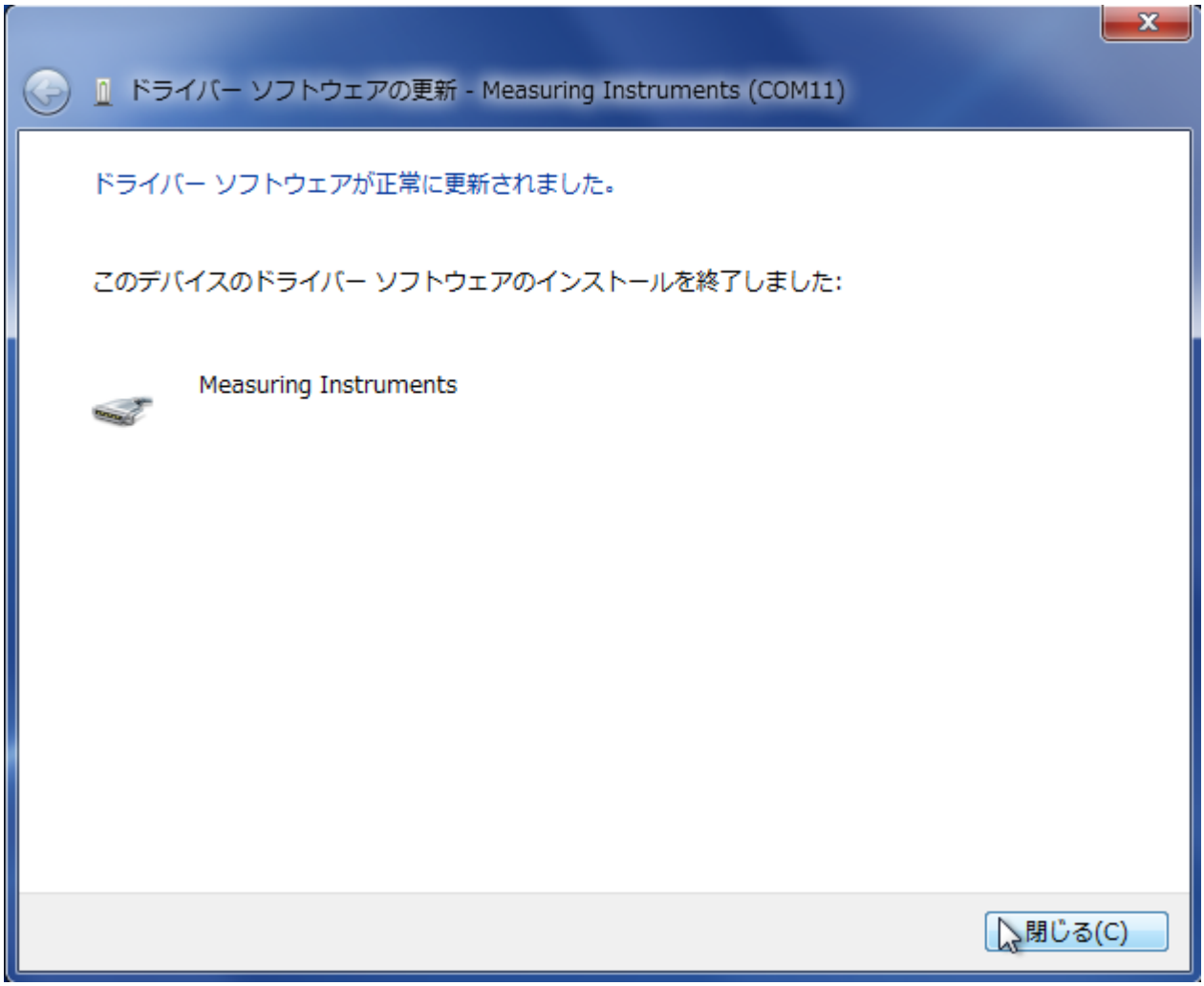
「次へ」をクリックします。



「インストール」をクリックします。

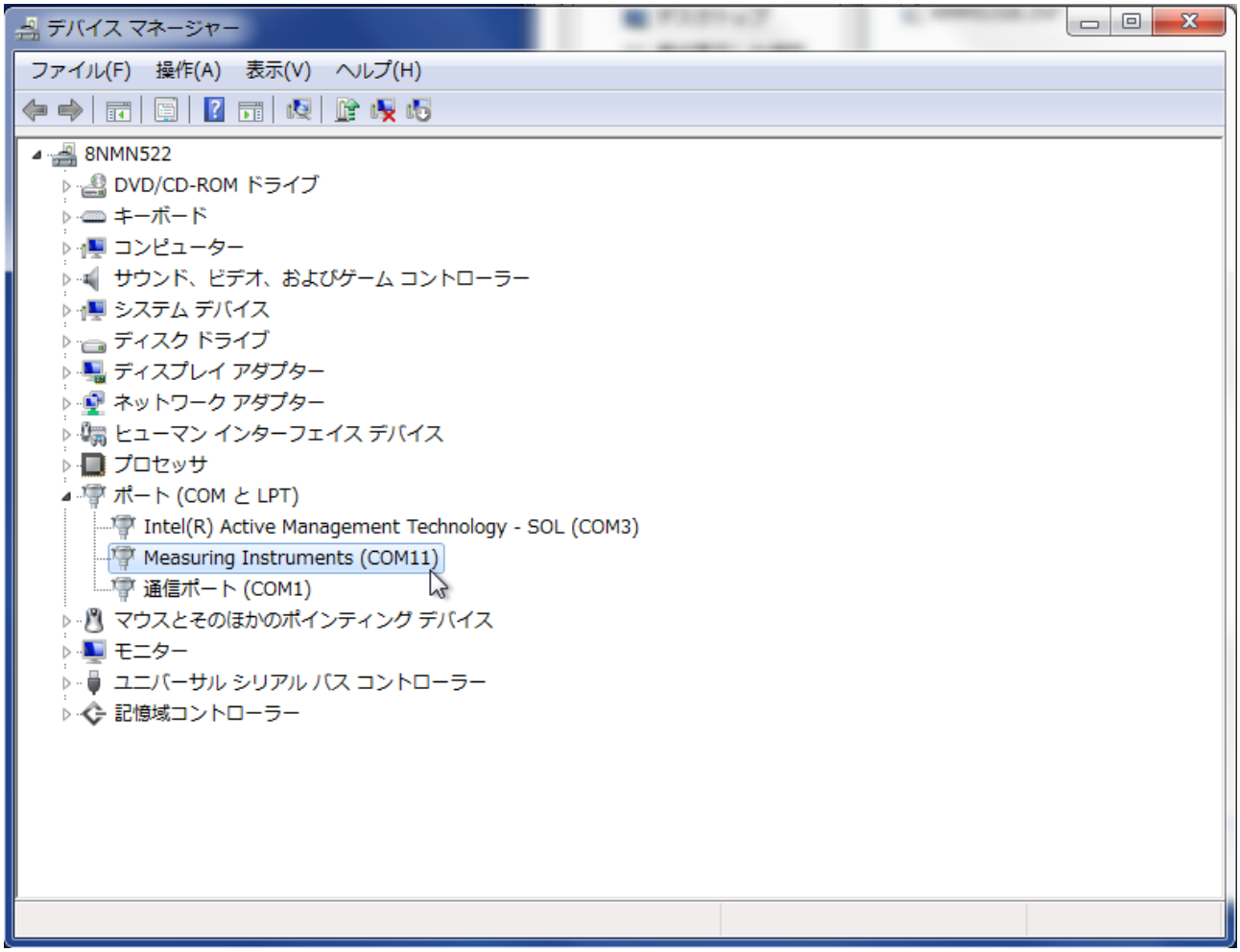


インストール終了を確認したら、閉じるをクリックします。



Measuring Instruments に警告マークが表示されておらず、続けて COM 番号が表示されていることを確認したら右上の×でウィンドウを閉じます。

(下図では COM11 となっていますが、この COM 番号は PC によって異なります。)



以上で、ドライバインストールは終了です。