PC Software for Color Analyzer CA-SDK Ver.4.10

Programming Guide

KONICA MINOLTA

●Note that this manual abbreviates product names as follows.

Name Given in This Manual	*Official Name*
VB, Visual Basic	Microsoft® Visual Basic
Windows	Microsoft® Windows®
Windows 98	Microsoft® Windows® 98 Operating System Second Edition
Windows Me	Microsoft® Windows® Millennium Edition Operating System
Windows 2000	Microsoft® Windows® 2000 Professional Operating System
Windows XP	Microsoft® Windows® XP Professional Operating System
Windows Vista	Microsoft® Windows® Vista Business Operating System

●Trademark Notices

- Microsoft and Windows are registered trademarks of the Microsoft Corporation, in the US and in other countries.
- Other company names and product names that appear within this manual are trademarks or registered trademarks of their respective companies.

●Notes on this Manual

- Copy or reproduction of all or any part of the contents of this manual without Konica Minolta's permission is strictly prohibited.
- The contents of this manual are subjects to change without prior notice.
- Every effort has been made in the preparation of this manual to ensure the accuracy of its contents. However, should you have any questions or find any errors, please contact a Konica Minolta authorized service facility.
- Konica Minolta will not accept any responsibility for consequences arising from the use of the software.

Contents

Foreword

The CA-SDK software development kit facilitates development of Windows-based applications for the CA-200 color analyzer ("CA"). This SDK makes use of Microsoft's COM (Component Object Model) architecture, but does not require developers to be knowledge about COM itself.
This manual assumes that developers are using Microsoft Visual Basic ("VB"). All programming examples are in VB.

CA-SDK controls CA-210 series and CA-100Plus in CA-200 mode. This manual describes all of the compatible models as "CA-200".
“Flicker measuring” is the function only for the CA-210 with LCD Flicker Measuring Probe, CA-P12/15 or Small LCD Flicker Measuring Probe, CA-PS12/15. All of description about “Flicker measuring” are applicable to only the CA-210 with these probes.

System Requirements

This SDK requires the following environment.

- DOS/V computer running the Windows 98, Windows ME, Windows 2000, Windows XP, or Windows Vista operating system.
- COM-compliant development tools (VB, etc.) installed on the computer.
- The computer must be equipped with an RS-232C port or a USB1.1-compliant USB port.

Notes on Use of SDK

When the PC connection is USB, running the application software by CA-SDK without CA-200 connected may cause to abort because of the runtime-error. In this case, please connect CA-200.

If the USB connecting cable is getting loose or disconnected while the application software is running, the application software will cause to abort because of the runtime-error, even if you connect it again while the application software is running. If the USB connecting cable gets disconnected and the USB connection error occurs, please end the application software first, then connect the USB connecting cable, and then restart the application software.

CA-SDK supports only “32-bits mode” of the application software. When you develop an application software, you need certify “32-bits mode”.

1. How to Install (or Uninstall) the SDK

If an older version of the SDK has been installed, you need to uninstall it before installing this version.

In the case of installing or uninstalling the SDK to the computer that is capable of autorun function

① Insert the SDK's Setup disk into the computer's CD-ROM drive.
The Setup Program (automatic installation) will start and now carry the installation.
Follow the instructions displayed on the screen.

Remarks: If the SDK already installed, automatic uninstallation will start.
If you don't wish to uninstall the SDK, please cancel the setup program.

In the case of installing the SDK to the computer that is not capable of autorun function, or failing automatic installation, try the manual operated installation as explaining below

① Insert the SDK's Setup disk into the computer's CD-ROM drive.
② Click the **Start** button, and select **Run**.
③ Click the **Browse** button. In the **Browse** dialog, set **Look in**: to the drive letter corresponding to the drive holding the Setup disk, and then select **Setup.exe** from the file list. Then click the **Open** button.
④ Confirm that **Setup.exe** is now shown in the **Open:** text box, and then click **OK** to start the installation.

Note: If you are connecting to the CA units by USB, you must also install the CA-200 USB driver. For information, refer to Section 6, "Installing the USB Driver."

In the case of uninstalling the SDK from the computer by the manual operation

If you wish to uninstall the SDK, use the general method for uninstalling a program (the Add/Remove applet in the Control Panel).

Additional information on installing/uninstalling the SDK under Windows 2000, Windows XP, or Windows Vista:

When installing or uninstalling the SDK on Windows 2000, Windows XP, or Windows Vista, be sure that you are logged in with Administrator rights.
If you are not sure of the current access rights, check with your system administrator.

Related points:
After installation, problems may occur if a user with limited rights attempts to change the pattern-generator settings when using the sample software to control a pattern generator. In this case, it is necessary to give the user suitable access permission for the SDK installation folder/files.

Setting example 1: The subject user should be given "Full control" access to the top-level SDK installation folder (the default top-level installation folder is normally "C:¥Program Files¥CA-SDK").

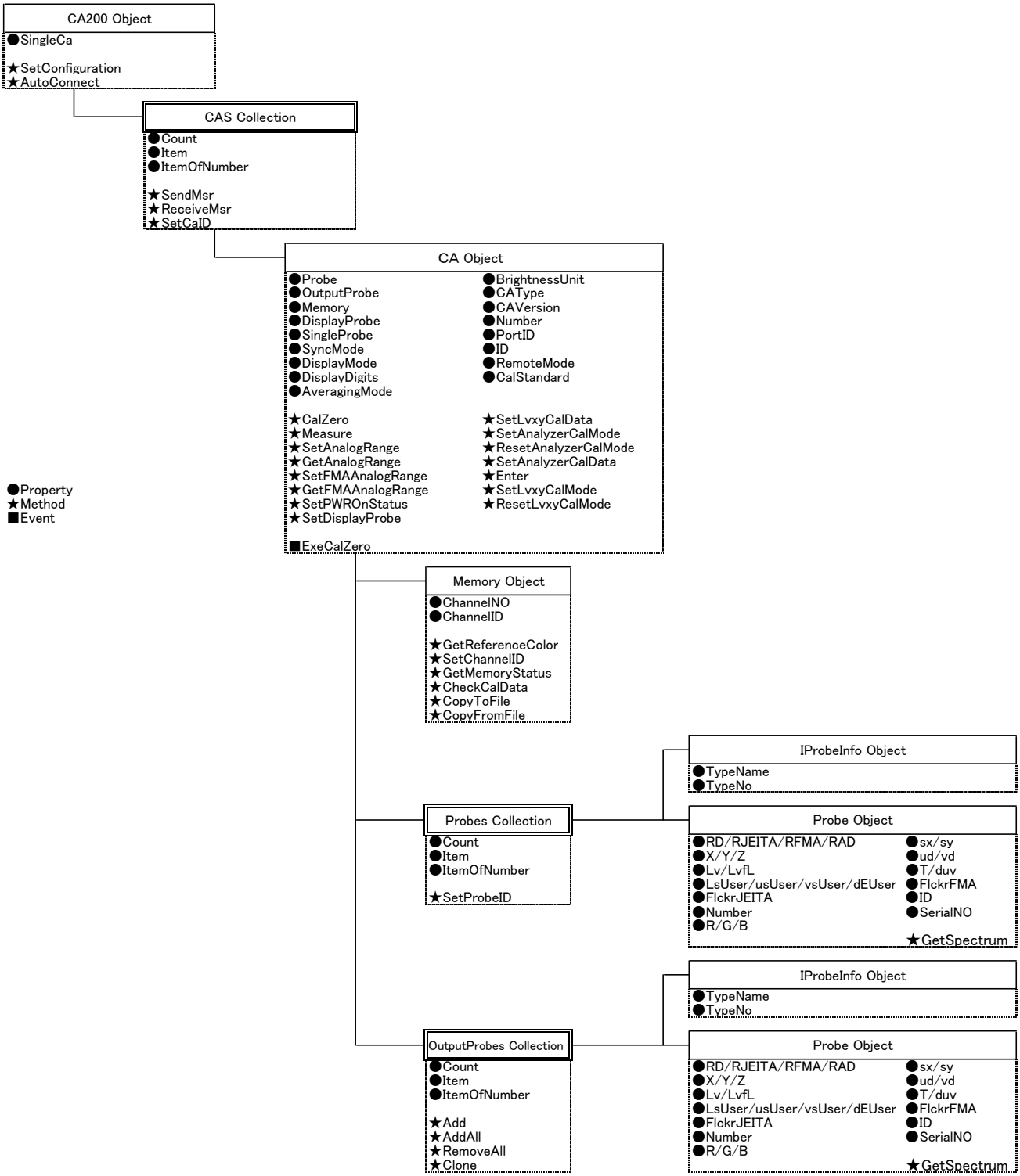
In addition, problems may occur if a restricted user attempts to use a development tool to open a sample software project and perform work.
If a problem does occur, after setting the above access permissions, perform the following sequence of works twice.

Sequence of works: Check that the project can be opened normally when using Administrator rights. If it can, close the project and restart the computer.
(Repeat this sequence twice.)

2. SDK Object Hierarchy

Object Hierarchy

This SDK exposes its capabilities by means of objects created in accordance with Microsoft's COM specifications. The objects exposed by this SDK implement a dual interface and support automation. The following diagram shows the hierarchy of the objects exposed by the SDK.



2.1 Object, Collection

`Ca200` Object

This is the SDK's application object. The `Ca200` object sets up the configuration of the connected CA-200 units. When you use the object's methods to set up the configuration, the software automatically instantiates and initializes the relevant lower-level objects (Cas level and below). These lower-level objects are then used to control the measurement system.
The `Ca200` object itself is the only object that is generated explicitly by the application.

`Cas` Collection

If you are using multiple CA-200 units, the `Cas` object is the collection of the Ca objects corresponding to these CA-200 units. The `Cas` collection is accessed as a property of the `Ca200` object. When you need to control a specific CA-200, you first must get that CA-200 using the appropriate method or property of the `Cas` collection.

`Ca` Object

Each `Ca` object controls a connected CA-200 unit. Specifically, this object exposes most of the physical CA-200 unit's control features. To control the CA-200, you use the properties and methods of the `Ca` object.

`Memory` Object

You use the `Memory` object to operate on the memory channels of the corresponding CA-200. The `Memory` object exposes the features supported through the CA-200 memory channels.

`Probes` Collection

This is a collection of the `Probe` Objects in the configuration where two or more probes are connected. The `Probes` Collection is obtained as `Ca` Object's Property. `Probe` Objects that are compliant with the connected probes are obtained using Property/Method of the `Probes` Collection.

`OutputProbes` Object

If you have multiple output probes connected to a single CA-200, the `OutputProbes` object is the collection of the Probe objects corresponding to these probes. The `OutputProbes` object is accessed as a property of the `Ca` object. To get the `Probe` object for a specific probe, you use the relevant property or method of the `OutputProbes` collection.
For any given CA-200, the `OutputProbes` object is the collection of the `Probe` objects corresponding to the probes that have been designated as the CA-200's output probes. The `OutputProbes` object is accessed as a property of the `Ca` object. You use the methods and properties of the `OutputProbes` object to get, add, and remove output probes.
While `Probe` objects can also be referenced as members of the `Probes` collection, when getting measurement results you should reference them as members of the `OutputProbes` collection.

`Probe` Object

Each probe on a CA-200 unit is represented by a `Probe` object. The `Probe` object serves as the container for the output probes results, and exposes these results to the application. To get measurement results, you use the properties and method of the `Probe` object.

`IProbeInfo` Object

This is a container of the hardware information for each probe of CA-200. The `IprobeInfo` Object provides the hardware information of CA-200. The Property of the `Probe` Object will be used for referring to the measurement results.

2.2 Method

Each Object/Collection includes Method (function).
The details of each Method are explained in 「4. SDK Reference」.

2.3 Property

Each Object/Collection includes Property.
Each Property has either both of the ① obtainment function and ② setting functions or one of the functions.
The details of each Property are explained in 「4. SDK Reference」.

2.4 Event

There is `ExeCalZero` among the `CA` Object. The details are explained in 「4. SDK Reference」.

【Reference】Overview of the SDK's Objects and Their Methods, Properties, and Collections

The following is a listing and brief description of the objects provided by the SDK. The listing shows the SDK objects and their methods, properties, and collections.

`Ca200` Object

Properties / Collections:

`Cas`	Collection of connected CA-200
`SingleCa`	Gets CA-200 that was set up using the `AutoConnect` method

Methods:

`SetConfiguration`	Sets the CA-200 configuration
`AutoConnect`	Automatically sets configuration to: USB connection, single CA-200, single probe

`Cas` Collection

Properties:

`Item`	Gets the designated CA-200 from the CA-200s collection (by index value or ID name)
`Count`	Gets the count of the connected CA-200s
`ItemOfNumber`	Gets the designated CA-200 from the CA-200s collection (by ID number)

Methods:

`SendMsr`	Sends the Measure command to all connected CA-200s
`ReceiveMsr`	Gets measurement results from all connected CA-200s
`SetCaID`	Sets the name to the designated CA-200

`Ca` Object

Properties / Collections:

`Probes`	Collection of probes connected to the CA-200
`OutputProbes`	Collection of those connected probes that are designated as output probes
`Memory`	Gets the CA-200's memory channels
`DisplayProbe`	Sets or gets the CA-200's display probe
`SingleProbe`	Gets probe that was configured using the `AutoConnect` method
`SyncMode`	Sets or gets the CA-200's sync mode
`DisplayMode`	Sets or gets the CA-200's display mode (measuring mode)
`DisplayDigits`	Sets or gets the display-digits settings (the number of digits used in the CA-200's display)
`AveragingMode`	Sets or gets the CA-200's averaging mode (measurement rate)
`BrightnessUnit`	Sets or gets the dimensional unit that the CA-200 uses for its brightness display
`CAType`	Gets the CA-200's product type
`CAVersion`	Gets the CA-200's product firmware version information
`Number`	Gets the CA-200's ID No.
`PortID`	Gets the CA-200's comm port ID
`ID`	Sets or gets the CA-200's ID name
`RemoteMode`	Sets the CA-200's Remote mode ON or OFF
`CalStandard`	Sets or gets the CA-200's Calibration Standard

Methods:

`CalZero`	Zero-calibrates the CA-200
`Measure`	Executes measurement
`SetAnalogRange`	Sets the CA-200's analog display range
`GetAnalogRange`	Gets the CA-200's analog display range
`SetFMAAnalogRange`	Sets the analog display range used by the CA-200 for flicker measurement（Only the applicable model）
`GetFMAAnalogRange`	Gets the analog display range used by the CA-200 for flicker measurement（Only the applicable model）
`SetPwrONStatus`	Establishes the CA-200's current settings as its power-on settings
`SetDisplayProbe`	Selects the probe whose results will be displayed on the CA-200's screen
`SetAnalyzerCalMode`	Sets the CA-200 to display-characteristics input mode
`ResetAnalyzerCalMode`	Switches the CA-200 from display-characteristics input mode back into normal mode
`SetAnalyzerCalData`	Executes input of calibration data (measurement data)
`Enter`	Writes arbitrary calibration data or display-characteristics data into memory
`SetLvxyCalMode`	Sets the CA-200 into arbitrary calibration mode
`ResetLvxyCalMode`	Switches the CA-200 from arbitrary calibration mode back into normal mode
`SetLvxyCalData`	Executes input of arbitrary calibration data (measured values and calibration data)

Event:

`ExeCalZero`	Notification that zero-calibration is required

`Memory` Object

Properties:

`ChannelNO`	Selects one of the CA-200's memory channels, or returns the current channel selection (where selection is expressed by the channel's connector number)
`ChannelID`	Selects one of the CA-200's memory channels, or returns the current channel selection (where selection is expressed by the channel's ID name)

Methods:

`GetReferenceColor`	Gets the reference color (white) setting of the selected memory channel
`SetChannelID`	Sets an ID name for the selected memory channel
`GetMemoryStatus`	Gets calibration information from the selected memory channel
`CheckCalData`	Compares the selected channel's calibration data against data in the calibration data file
`CopyToFile`	Copies the selected memory channel's calibration data to file
`CopyFromFile`	Copies data from the calibration data file into the selected memory channel

`Probes` Collection

Properties:

`Item`	Gets the designated probe from the `Probes` collection (by index value or ID name)
`Count`	Gets the count of the connected probes
`ItemOfNumber`	Gets the designated probe from the `Probes` collection (by ID number)

Method:

`SetProbeID`	Sets an ID name (alias) for a specified `Probe` object

`Probe` Object

Properties:

`X`, `Y`, `Z`	Gets measurement result, as represented in **XYZ** color space
`Lv`, `LvfL`	Gets brightness measurement result, in indicated units
`sx`, `sy`	Gets measurement result, as represented in **xy** color space
`ud`, `vd`	Gets measurement result, as represented in **Lu'v'** color space
`T`, `duv`	Gets correlated color temperature or difference from black-body locus, as represented in **uv** color space
`R`, `G`, `B`	Gets analyzer-mode measurement results
`LsUser`, `usUser`, `vsUser`	Gets measurement result, as represented in **L*u*v*** color space
`FlckrJEITA`	Gets flicker quantity, as measured using the JEITA method.[*1] [Available only on CA models that support this feature.]
`FlckrFMA`	Gets flicker quantity, as measured using the FMA method (AC/DC method).[*2] [Available only on CA models that support this feature.]
`Number`	Gets the probe's ID number
`ID`	Sets or gets the probe's ID name
`SerialNO`	Gets the probe's serial number
`RD/RJEITA/RFMA/RAD`	Gets color value, flicker quantity as measured using the JEITA method[*1] , flicker quantity as measured using the FMA method (AC/DC method)[*2] , and status code of analyzer-mode measurement [Available only on CA models that support this feature.]
`GetSpectrum`	Gets the frequency components of flicker as measured using the JEITA method[*1] [Available only on CA models that support this feature.]

[*]1. JEITA flicker method
[*]2. Contrast flicker method

`OutputProbes` Collection

Properties:

`Item`	Gets the designated probe from the `OutputProbes` collection (by index value or ID name)
`Count`	Gets the count of the connected output probes
`ItemOfNumber`	Gets the designated probe from the `OutputProbes` collection (by ID number)

Method:

`Add`	Establishes the specified probe as an output probe
`AddAll`	Establishes all connected probes as output probes
`RemoveAll`	Removes all output probes
`Clone`	Gets copy of `OutputProbes` collection

`IProbeInfo` Object

Properties:

`TypeName`	Obtains a character string that indicates the type of the connected probes
`TypeNO`	Obtains a value that indicates the type of the connected probes

3. Writing Programs with the SDK

The sample code shown in this section includes a number of symbolic constants that are not explicitly explained. For details about parameter usage in methods and properties, see Section 4, "SDK Reference."

3.1 Basics

3.1.1 Using the SDK to Create a Program: Overview（Single Probe）

Basic procedures and examples of simple programming when the number of main body: 1, the number of Probe: 1 and PC connection: USB are shown below (Please refer to each page for the details).
Please make sure to execute ①～④, which are processing necessary for sound recognition on the program.

1. Each program comprises the following steps.

①Application Setup (CA200Srvr Type Library) (Please refer to P. 11)
②Declare the SDK object variables (variables that represent the actual measuring instruments and probes). (Please refer to P. 13)
③Set the CA-200 configuration (Number of CA-200, and number of connected probes). (Please refer to P. 14)
④Set up the CA-200.'Create object variables. (Please refer to P. 15)
⑤Set the measurement conditions. (Please refer to P. 15)
⑥Execute measurement. (Please refer to P. 17)
⑦Get the measurement results. (Please refer to P. 17)
⑧Process the measurement results.
⑨Calibration . (Please refer to P. 21)
⑩Carry out relevant error processing. (Please refer to P. 29)

2. Example of a Simple Program (for a single probe)

②Declare the SDK object variables.

```
Public objCa200 As Ca200          'Application object
Public objCa As Ca                'CA-200 object
Public objProbe As Probe          'Probe object
Public objMemory As Memory        'Memory object

On Error GoTo Er                  'Set the error trap
```

③Set the CA-200 configuration.

```
'Create the application object.
Set objCa200 = New Ca200

'Set the configuration (automatic configuration).
objCa200.AutoConnect              '1 CA-200, 1 probe, USB connection
```

④Set up the CA-200.

```
'Create object variables.
Set objCa = objCa200.SingleCa  'CA-200 object
Set objProbe = objCa.SingleProbe        'Probe object
Set objMemory = objCa.Memory   'Memory object

'Send zero-calibration start message to operator. (Code is omitted.)

'Run zero-calibration.
objCa. CalZero
```

⑤'Set the measurement conditions.

```
objCa.SyncMode = 3                  'Set sync mode to UNIV sync.
objCa.AveragingMode = 2             'Set measuring rate to AUTO.
objCa.SetAnalogRange 2.5, 2.5       'Set analog display range to 2.5% units.
objCa.DisplayMode = 0               'Set display/measurement mode to Lvxy.
objMemory.ChannelNO = 0             'Set memory channel to 0 (Konica Minolta calibration).
```

⑥Execute measurement.

```
'Send start message. (Code is omitted.)

'Execute measurement.
objCa.Measure
```

⑦Retrieve the measured data.

```
'Declare the measurement data variables.
Dim fLv As Single
Dim fx As Single
Dim fy As Single

'Load measurement values into measurement data variables.
fLv = objProbe.Lv
fx  = objProbe.sx
fy  = objProbe.sy
```

⑧Process the measurement data.

```
'Data display processing, etc.    (Code is omitted.)
```

⑨Calibration.

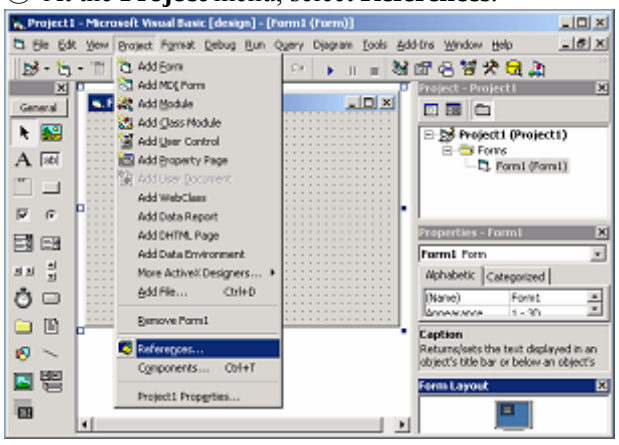
```
 'Processing at time of calibration    (Please refer to P. 22.)
```

⑩Carry out relevant error processing.

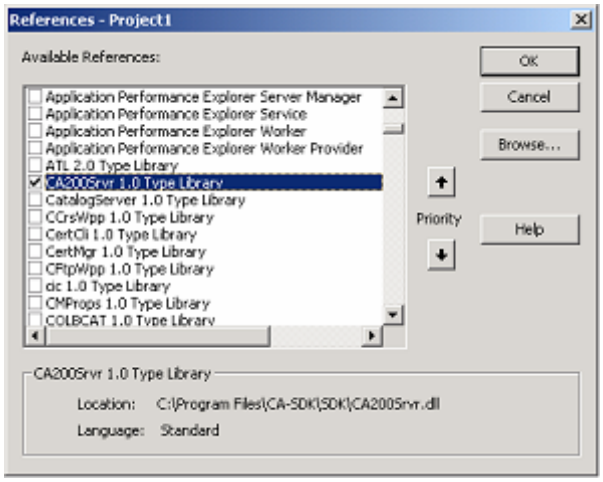
```
 'Processing at time of error    (Please refer to P. 27.)
 Er:
     Dim strERR As String
     Dim iReturn As Integer

     strERR = "Error from " + Err.Source + Chr$(10) + Chr$(13)
     strERR = strERR + Err.Description + Chr$(10) + Chr$(13)
     strERR = strERR + "HRESULT " + CStr(Err.Number - vbObjectError)
     iReturn = MsgBox(strERR, vbRetryCancel)
     Select Case iReturn
         Case vbRetry: Resume
         Case Else:
             objCa.RemoteMode = 0
             End
     End Select
```

3.1.2 Application Setup (CA200Srvr Type Library)

① Start up VB, and create the project to be used for the application development.
② At the **Project** menu, select **References**.



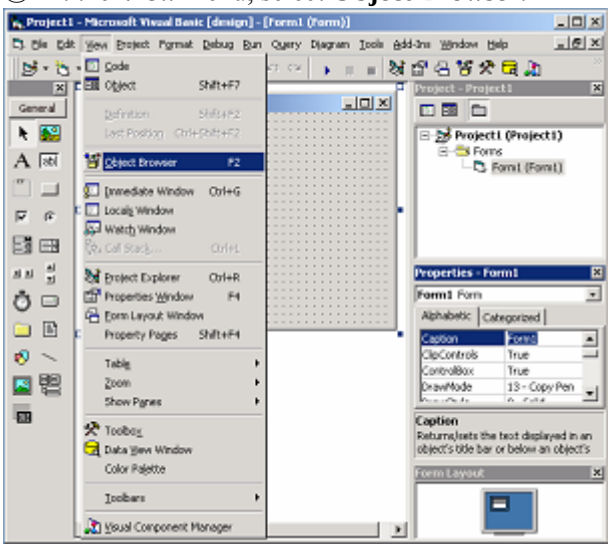
③ In the **Available References** list, place a check next to **CA200Srvr 1.0 Type Library**. Then click the **OK** button.



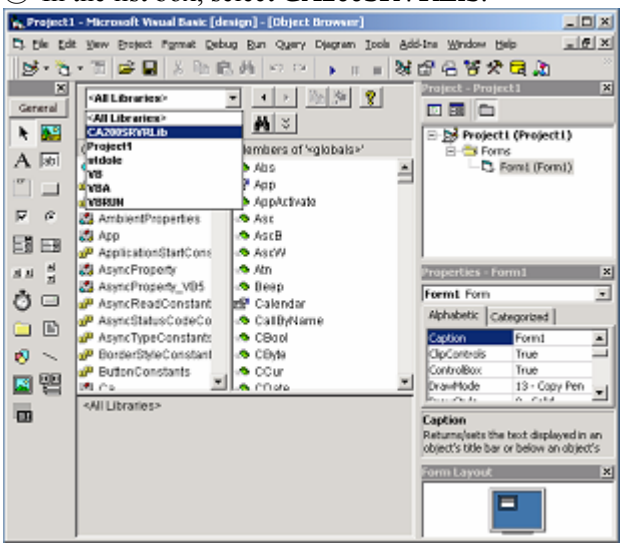
The application will now be able to utilize the SDK functions.

You can also use the VB object browser to view SDK object information, by proceeding as follows.

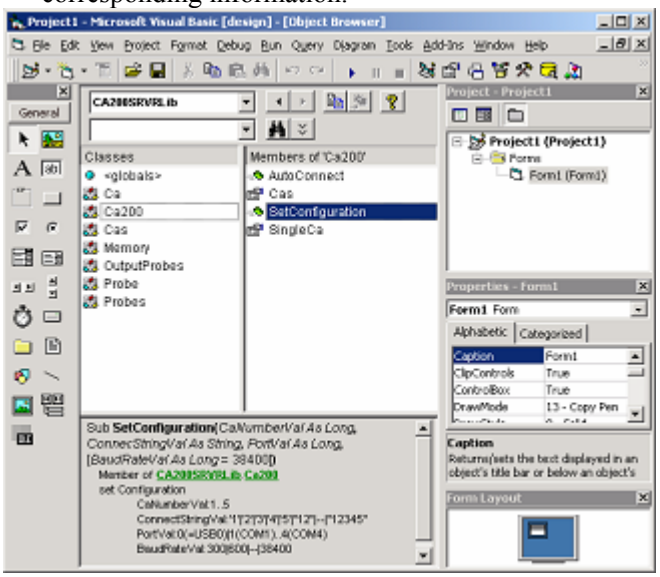
① At the **View** menu, select **Object Browser**.



② In the list box, select **CA200SRVRLib**.



③ In the Classes and Members viewing areas, click the relevant Class or Member entry to view the corresponding information.



3.1.3 Set up the CA-200(Create object variables)

This section presents an overview of the SDK's features and usage. For related information, refer also to Section 2, "SDK Object Hierarchy."

In VB programming, the programmer manipulates VB-specific objects by using the properties and methods exposed by these objects. Programming with the SDK follows the same principle: the SDK exposes various objects, and the programmer uses VB to manipulate these objects by means of the properties and methods exposed by these objects.
The objects exposed by this SDK are designed to model the configuration, operation, and environment of a physical CA-200 setup. If you understand how to use the CA-200 hardware, usage of the SDK objects should be very straightforward.
You begin a program by instantiating SDK objects (object variables) so as to create a representation of the CA environment that you are working with. Once defined, these variables will remain in existence and available throughout the life of the program. Accordingly, these variables must be declared as globals.

Example: Declaration of SDK Object Variables

```
Public objCa200 As Ca200      'Application object
Public objCa As Ca            'CA-200 (instrument) object
Public objProbe As Probe      ' Probe object
Public objMemory As Memory    ' Memory object
```

Please note the following points.

- This SDK operates as a COM-compliant in-process server (produced in accordance with Microsoft's COM standard). The SDK is packaged in DLL form, and cannot be used as standalone software.
- This SDK is designed on the premise that it will be used from a single client. It is not designed for multithread usage, or for use with Microsoft's MTS.
- Although the SDK is COM-compliant, it is assumed that the programmer will use the SDK objects in accordance with the object hierarchy explained in Section 2, "SDK Object Hierarchy."

3.1.4 Setting Up and Managing the CA-200 Configuration

This section explains how to write program code to set up and manage CA-200 configurations for a variety of usage environments and objectives.

Usage environments and objectives can vary significantly. In some cases you can meet your needs using a single CA-200 unit, while in other cases you will need to control multiple CAs. Similarly, you may or may not need to control multiple probes on each CA-200. (Each CA-200 can connect up to five probes.) Accordingly, your program must set up the appropriate CA configuration—specifying how many CA-200s to use, and how many probes to connect up to each. Similarly, the appropriate type of management to carry out with respect to this connected hardware will also vary according to your specific objectives.

To configure the CA-200s, you use one of the following two methods of the `CA200` object.

- `AutoConnect` method
- `SetConfiguration` method

The `AutoConnect` method is for straightforward, simple setups, and can only be utilized when you are using a single CA-200 with a single probe, and where the connection between the computer and the CA-200 is by USB. When you use this method, the SDK will detect the connection status and automatically set up the appropriate configuration.

If you need to manage multiple CA-200s and/or multiple probes, then you must use the `SetConfiguration` method to explicitly set the configuration. When using this method, you explicitly set up and manage the CA-200 connection information and the probe connection information. If the `SetConfiguration` method detects that the actual hardware configuration conflicts with your configuration setup information, it will terminate with an error.

The following examples show how to use the `SetConfiguration` method to connect up two different configurations. In the first example, the method is used to configure a single CA-200 with five probes, using USB connection. In the second example, the method sets up three CA-200s, each with multiple probes. (For detailed information about the `SetConfiguration` method, refer to explanation of this method in Section 4, "SDK Reference", below.)

Example 1: USB connecting one CA-200 with five probes

```
' Create the application object.
Set objCa200 = New Ca200

' Set the configuration.
objCa200.SetConfiguration 1, "12345", 0      'USB
```

Note that the method sets the CA's ID No. to 1—which means that it sets the `CAID` property of the instantiated `Ca` object to "CA1". The method also sets up connection of five probes to be connected to probe numbers 1 to 5. It does this by instantiating five `Probe` objects, each with a corresponding `ID` property value (from "P1" to "P5").

Example 2: COM1, COM2, or COM3 connecting three CAs, each having four probes

```
' Create the application object.
Set objCa200 = New Ca200

' Set the configuration.
objCa200.SetConfiguration 1, "1234", 1 ,38400      'COM1
objCa200.SetConfiguration 2, "1234", 2 ,38400      'COM2
objCa200.SetConfiguration 3, "1234", 3 ,38400      'COM3
```

In this example, the method instantiates three `Ca` objects—one for each of the three CAs. Consider the first CA (the CA connected to COM1). The method assigns ID No. 1 to this CA—which means that it sets the `CAID` property of the corresponding `Ca` object to "CA1". It also specifies that four measurement probes are to be connected to CA probe connectors 1 to 4—which means that it instantiates four `Probe` objects for this `Ca` object, and sets the `ID` property for the first `Probe` objects to "P1", for the second to "P2", and so on. The method sets up the other two `Ca` objects in similar fashion.

Further, all three of these instantiated `Ca` objects are automatically added as members to the `Cas` collection; and within each `Ca` object, each instantiated `Probe` object is automatically added to the `Probes` collection. The generated `CAID` and probe `ID` property values can then be used to target and operate on the relevant objects within these collections (as explained starting from Section 3.1.6, below.)
Provided that the configuration information set by the method matches the actual hardware configuration, the method's setup operation terminates normally and the program can thereafter use the various objects to control the operation of the CAs and probes.

3.1.5 Assignment of the object variables (generation)

An assignment of the object variables always need to be done after executing 「3.1.3 Declaration of the SDK object variables」 and 「3.1.4 Generation of the SDK application object and setting the configuration」.

```
'Generating objects to object variables.
Set objCa = objCa200.SingleCa          'CA-200 objet
Set objProbe = objCa.SingleProbe       'Probe object
Set objMemory = objCa.Memory           'Memory object
```

3.1.6 Set the measurement conditions

This section explains how to use write code to set up a CA-200 unit in preparation for taking measurements.

Before executing measurement, you need to set each CA-200 so that it will carry out the appropriate type of measurement operation. Setup involves three basic steps, as follows.

① Zero-calibration

Zero-calibration must be carried out on the CA-200 after the CA-200 has stabilized. Attempts to take measurements with a CA-200 that is not zero-calibrated will return an error.

② Setting of general measurement parameters (permanent and semi-permanent settings)

- Measurement Sync Mode
- Measurement Speed
- Brightness Display Unit
- Number of Display Digits
- Analog Range Setting (for color or flicker measurement)
- Calibration channel (the default [Konica Minolta] calibration standard)

When using the SDK to set up a CA-200, you use the `Ca` object's zero-calibration method to run zero calibration; and you use the object's various properties to enter the relevant settings. The following table shows the correspondence.

Action or Setting	Corresponding method or property of `Ca` object
Zero Calibration	`CalZero( )`
Measurement Sync Mode	`SyncMode`
FAST/SLOW Mode	`AveragingMode`
Brightness Unit	`BrightnessUnit`
Number of Display Digits	`DisplayDigits`
Analog Range (for color measurement)	`SetAnalogRange( )`
Analog Range (for flicker measurement)	`SetFMAAnalogRange( )`
Default (Konica Minolta) calibration	`CalStandard`

The following example shows how to carry out the above-described setup. The example begins by zero-calibrating the CA-200 (using the `Ca` object's `CalZero` method). It then uses the object properties shown above to set the CA-200 as follows: use NTSC sync; use Auto measurement speed; use cd/m^2 display unit; use 3-digit display; use 9300K calibration standard.

Note that the example does not set the Analog Range. The Analog Range setting is required only if you are using the CA-200 unit's analog display.

Example: Setting Up a CA-200

```
' Set up the CA-200.

' ① Run zero-calibration.
objCa.CalZero

' ② Enter (semi-)permanent parameter settings.
With objCa
   .SyncMode = 0           'Set to NTSC sync.
   .AveragingMode = 2      'Set measurement rate to Auto.
   .BrightnessUnit = 1     'Set brightness display unit to cd/m2.
   .DisplayDigits = 0      'Set to 3-digit display.
   .CalStandard = 2        'Set to 9300K calibration standard.
End With
```

③ One-Time Settings

For information about settings that tend to change for each measurement cycle, refer to the section immediately below.

3.1.7 Executing Measurement and Getting the Results

This section explains how to write program code to execute measurements and retrieve the measurement results.

A number of settings tend to change frequently with each measurement cycle. Before taking measurement, you need to change these settings as necessary. Typical one-time settings include the following.

- What to measure? … Set the display mode (measurement mode).
- Where to measure? … Select the output probe.
- Which calibration data to use? … Select the memory channel.

If you intend to use the CA-200's own analog display, then you also need to set the following:

- Display data from which probe? … Select the display probe.

Setting the Display Mode (Measurement Mode)

The CA-200 can measure and display a variety of items. The two main measurement types are:

① Measurement of color/brightness
② Analyzer-mode measurement

In addition, some CA models also support the following:

③ Flicker measurement

When measuring color or brightness, you can select from a variety of available display methods (color spaces). When using analyzer-mode measurement (a Konica Minolta proprietary feature that enables simple white adjustment), you can select from three display methods. For flicker measurements, you can choose from two measurement methods.

The selection of the display mode (measurement mode) determines the type of measurement that the CA-200 will carry out, and also determines which of the `Probe` object properties will be updated with the measurement results. The following table shows the relationships.

Measurement Type	Display Mode	Updated properties in the `Probe` object
Color / Brightness	XYZ (Value in **XYZ** color space)	`X`, `Y`, `Z`, `Lv`, `sx`. `sy`, `ud`, `vd`, `T`, `LsUser`, `usUser`, `vsUser`, `dEUser`, `duv`
	xyLv (Chromaticity in **Lvxy** color space)	
	L*u*v* color-space chromaticity	
	T⊿uv (corrected color temp. and divergence from black-body locus, in **uv** color space)	
	u'v'Lv (Chromaticity in **u'v'** color space)	
Analyzer Mode	No analog display	`R`, `G`, `B`
	G-based analog display	
	R-based analog display	
Flicker*3	AC/DC-method flicker value	`FlckrFMA`
	JEITA-method flicker value*1	`FlckrJEITA`

*1: For JEITA flicker measurement. results are not displayed on the CA-200 unit itself.
*2: The CA-200 unit does not display **L*u*v*** chromaticity or color difference.
*3: Only the applicable model
When using the SDK, you set the display mode (measurement mode) by setting the `Ca` object's `DisplayMode` property.

Specifying the Output Probes

An *output probe* is a probe that has been set to output its measurement results data to the PC through a communications channel. If you wish to execute control and data processing from the PC, you need to designate the relevant probes as output probes. The output probe setup also affects the speed with which the CA-200 transmits measurement responses to the PC: if you are carrying out repetitive measurement and you need rapid response, you can speed up the response time by limiting output probes to those that you need to work with. (If you have used the `SetConfiguration` method to set the configuration, you must also designate output probes if you wish to output the measurement results.)

To designate probes as output probes and to manage such designations, you use the properties and methods of the `OutputProbes` object.

Specifying the Memory Channel

When using the SDK, you select the memory channel by setting the `ChannelNO` or `ChannelID` property of the `Ca` object's `Memory` object. Setting the appropriate value into either of these properties will implement the actual setting on the targeted CA-200.

Specifying the Display Probe

The CA-200 unit itself can display results from a single probe only. If you are using multiple probes, therefore, you use the *display probe* setting to designate which of the probes is to be used for the display. When using the SDK, you can specify the display probe using the `Ca` object's `DisplayProbe` property.

The display probe setting also serves another purpose: it designates the probe that is calibrated when you run a calibration. For information about calibration, see Section3.5, "Managing CA-200 Calibration and CA-200 Calibration Data."

After setting the one-time measurement parameters, you execute measurement using the `Measure` method of the `Ca` object. Once measurement has been taken, the `Ca` object will automatically get the results by communicating directly with the CA-200 via the communication channel, and these results will then be written into the relevant `Probe` object properties, where they will be available to the application. (For a listing of results properties of the `Probe` object, refer to the `Probe` object listing in Section 2 above.)

Note that when you run the `Measure` method, only the relevant `Probe` object properties (the properties corresponding to the currently selected display type) will be updated with the new results. Unrelated properties retain the values they had before the method was executed.

In the programming example below, the program first executes the Zero-Calibration and then uses the `DisplayMode` property to set the measurement mode to Lvxy. It then uses the `Add` method of the `OutputProbes` collection to add three probes of " P1", " P2", and " P3"(the default ID names for probe number 1, 2, and 3), as `Probe` objects, to the collection, thereby establishing the probes with probe number 1, 2, and 3 as the CA-200's output probes. Next it uses the `ChannelNO` property of the `Memory` object to 0 to select use of the default calibration data. The CA-200 is now set to measure color and brightness, using its default calibration data for the calibration.

The program then executes the measurement by executing the `Measure` method.

Next, the program uses the `Item` property of the `OutputProbes` collection to designate the " P1", " P2", and " P3" probe objects. It then gets results from those probes, using each `Probe` object's `X`, `Y`, and `Z` properties get the XYZ tricolor stimulus values, and the `x` and `y` properties to get the Lvxy-space x and y values.

Next, the program uses the `RemoveAll` method of the `OutputProbes` collection to remove all `Probe` objects from the collection. It then changes the `DisplayMode` property to JEITA flicker display. Next, it uses the `Add` method of the `OutputProbes` collection to add the " P2" probe's `Probe` object to the collection—this time selecting only a single probe as the output probe. The code proceeds to execute JEITA flicker measurement and to get the flicker measurement results from the " P2" probe. (Since JEITA flicker measure generates a high communications data load, when taking repetitive flicker measurements you may want to limit the number of output probes so as to increase the measurement rate.)

Example: Program to Carry Out Measurement

```
'Generate application objects.
Set objCa200 = New Ca200

'Set the configuration.
objCa200.SetConfiguration 1, "123", 0 ,38400
                                        'USB, 3 probes with probe number 1, 2, and 3.

'Get the CA object.
Set objCa = objCa200.Cas.ItemOfNumber(1)

'Execute Zero-Calibration.
objCa.CalZero

'Measuring Routine
'   Set one-time measurement parameters.
'   Set the display mode (measurement mode).

objCa.DisplayMode = 0                   'Set measurement mode to Lvxy.

'   Select the output probes.

objCa.OutputProbes.Add("P1")            ' Add " P1" as output probe.
objCa.OutputProbes.Add("P2")            ' Add " P2" as output probe.
objCa.OutputProbes.Add("P3")            ' Add " P3" as output probe.

'   Set the memory channel.
'   Use the memory channel having the default calibration data (channel 0).

objCa.Memory.ChannelNO = 0

'   Execute measurement.

objCa.Measure

'   Get measurement results data.
'   Get XYZ data.

fX_R = objCa.OutputProbes.Item("P1").X
fY_R = objCa.OutputProbes.Item ("P1").Y
fZ_R = objCa.OutputProbes.Item ("P1").Z
fX_C = objCa.OutputProbes.Item("P2").X
fY_C = objCa.OutputProbes.Item ("P2").Y
fZ_C = objCa.OutputProbes.Item ("P2").Z
fX_L = objCa.OutputProbes.Item("P3").X
fY_L = objCa.OutputProbes.Item ("P3").Y
fZ_L = objCa.OutputProbes.Item ("P3").Z

'   Get xy data.

fsx_R = objCa.OutputProbes.Item ("P1").sx
fsy_R = objCa.OutputProbes.Item ("P1").sy
fsx_C = objCa.OutputProbes.Item ("P2").sx
fsy_C = objCa.OutputProbes.Item ("P2").sy
fsx_L = objCa.OutputProbes.Item ("P3").sx
fsy_L = objCa.OutputProbes.Item ("P3").sy

'   Change settings of one-time measurement parameters.
'   Set the display mode (measurement mode).

objCa.DisplayMode = 8                   'Set measurement mode to JEITA flicker.

'   Set the output probes.

  objCa.OutputProbes.RemoveAll
  objCa.OutputProbes.Add("P2")
```

```
' Execute measurement.
objCa.Measure

  ' Get measurement results data.
  ' Get JEITA flicker data.
   fFlckrJEITA = objCa.OutputProbes ("P2").FlckrJEITA
```

Note: About Index Values and ID Numbers

This SDK exposes three collections: `Cas`, `Probes`, and `OutputProbes`. For each collection, you use the collection's `Item` property to refer to a specific object within that collection. The `Item` property can take either of two arguments:

① The ID name of the targeted object (the `CA`, `Probe`, or `OutputProbe` object)
② The index value assigned to the object when it was placed in the collection

Note particularly that the index value is not identical to the object's ID number. For example, assume that you set two probes "P2" and "P3" into the `OutputProbes` collection—where "P2" and "P3" are the ID names for the probes, and the corresponding ID numbers are 2 and 3. But notice that while the ID numbers for these probes are 2 and 3, the collection's index values for these probes will be 1 and 2.

ID numbers are fixed at the time the configuration is set up, and remain the same thereafter. Index values, however, change according to the way objects are added to the collection.

3.1.8 Multi-Unit USB Connection

1. USB port number and response of connected CA units

* The CA-SDK recognizes USB ports with integer numbers 0 to 4. When only one CA unit is connected, the USB port number is 0; when multiple units are connected, the port numbers are assigned in order from the lowest-numbered port number of the USB hub.
* These numbers are assigned at the time connection to a CA unit is performed. If, when multiple CA units are connected, an additional CA unit is connected, the port numbers will be reassigned in order from the lowest-numbered port number of the USB hub.
* Because of this, when using an application, the CA units which will be controlled should be arranged in a suitable configuration and switched on so that USB connection is completed before starting the application. If an additional CA unit is connected after the application has been started, because of the reason given above, the USB connection conditions will change, which could lead to problems with USB communication. Please avoid this situation.

2. Measurement speed with multiple CA units connected

* The CA-SDK provides two ways to perform measurements with multiple CA units when multiple CA units are connected.

① Sequential execution of the `Ca::Measure` method
② Execution of the `Cas::SendMsr` and `Cas::ReceiveMsr` methods

* For ①, the CA-SDK sends the measurement command to the CA unit, receives the measurement results, and then returns from the method. Because of this, if the measurement speed was 15 times/second when using a single CA unit, the measurement time would become approximately 7.5 times/second (15/2) when taking simultaneous measurements using 2 CA units.
* For ②, the CA-SDK uses the `Cas::SendMsr` method to first send the measurement command to all CA units. Then, the measurement results are received from each CA unit using the `Cas::ReceiveMsr` method. Since the time required for sending the commands is much shorter compared to the time required by the CA unit for measurements, all CA units will have started measurements before the measurement results from the first CA unit are received. In addition, since at about the time the measurement results have been

received from the first CA unit, the measurement results for the next CA unit will be ready, the overhead for simultaneous measurements using multiple CA units is much shorter than for ①.

* However, for ②, attention must be paid to the following points. The premise for communication with the CA units is the pair of sending command and receiving the execution results. For CA-SDK methods other than the `Cas::SendMsr` method and the `Cas::ReceiveMsr` method, execution of the method performs the pair of sending command and receiving results, but for these two methods, executing the `Cas::SendMsr` method sends the command, and executing the `Cas::ReceiveMsr` method receives the results of the sent command. Because of this, it is necessary to always execute these two methods in pairs. (The CA-SDK does not check if these methods are paired.) Because of this, the application should be constructed so that even if an error occurs when executing the `Cas::SendMsr` method, the `Cas::ReceiveMsr` method is always executed. (Please refer to 3. Sample Program below.)

3. Sample program

Example: 2 CA units connected via USB, with 5 probes connected to each CA unit

```
Dim objCa200 As Ca200
Dim objCas As Cas
Dim objCa1 as Ca
Dim objCa2 as Ca

On Error Goto Err

Set objCa200 = New Ca200

' Setting 2 CA units, 5 probes per unit, USB connection.
objCa200.SetConfiguration 1, "12345", 0, 38400
objCa200.SetConfiguration 2, "12345", 1, 38400
Set objCas = objCa200.Cas
Set ObjCa1 = objCas.ItemOfNumber(1)
Set ObjCa2 = objCas.ItemOfNumber(2)
objCa1.OutputProbes.AddAll
objCa1.OutputProbes.AddAll

' Execute zero calibration.
objCa1.CalZero
objCa2.CalZero

' Execute measurement with all CA units, and receive measurement results.
objCas.SendMsr
objCas. ReceiveMsr

（Code is omitted.）

Err:

' Even if error occurs for SendMsr, ReceiveMsr must be executed to complete the pair.
Resume Next
```

3.2 Applications

3.2.1 Managing CA-200 Calibration and CA-200 Calibration Data

This section explains how to use the SDK to calibrate CA-200s and to manage the calibration data.

CA-200 units are capable of three types of measurement: color/brightness measurement, analyzer-mode measurement, and flicker measurement. Calibration for the color/brightness and analyzer-mode measurements can be set up as follows.

For color/brightness measurement, the user can set user standard values for the color and brightness of the target display, and then use these values as the basis for calibrating the CA-200. Within this manual, this type of calibration is referred to as *arbitrary calibration*. One of the arbitrary calibration is performing to each color of R, G, B, and W. In this manual, it is referred to as *matrix calibration*. The other of arbitrary calibration is performing to only W. In this manual, it is referred to as *white calibration*.
For analyzer-mode measurement, the CA-200 can be calibrated by writing into its memory standard R, G, and B values obtained by measuring the target display in some standard operating state (white-adjusted state, etc.). In this manual, this is referred to as *display-characteristics input*.
The SDK supports among matrix calibration, white calibration, and display-characteristics input. The following table shows the steps required for carrying out matrix and white calibration, and the support provided by the SDK.

Step	CA-200	SDK
1	Set connector number of probe to be calibrated.	`Ca` object `DisplayProbe` property
2	Set display mode to Lvxy.	`Ca` object `DisplayMode` property
3	Set calibration channel number.	`Memory` object `ChannelNO` property
4	Enter arbitrary calibration mode.	`Ca` object `SetLvxyCalMode` property
5	Display R, G., B, or W calibration pattern.	
6	Execute measurement.	`Ca` object `Measure` method
7	Input arbitrary calibration data. Specifically, enter measurement results and Lv, x, and y settings (user standard values).	`Ca` object `SetLvxyCalData` method
8	Write arbitrary calibration data to memory.	`Ca` object `Enter` method

For matrix calibration, repeat steps 5 to 7 for each color R, G, B, and W, and then carry out step 8 to write the results to memory. For white calibration, carry out the above procedure for W only.

The next table shows the steps required for carrying out display-characteristics input, and the support provided for these operations by the SDK.

Step	CA-200	SDK
1	Set connector number of probe to be calibrated.	`Ca` object `DisplayProbe` property
	Set display mode to analyzer mode.	`Ca` object `DisplayMode` property
2	Set calibration channel number.	`Memory` object `ChannelNO` property
3	Enter analyzer calibration input mode.	`Ca` object `SetAnalyzerCalMode` method
4	Display R, G, B, or W calibration pattern.	
5	Execute measurement.	`Ca` object `Measure` method
6	Input measurement results as display-characteristics values.	`Ca` object `SetAnalyzerCalData` method
7	Write arbitrary calibration data to memory.	`Ca` object `Enter` method

To complete the input, you must repeat steps 4 to 6 for each pattern (R, G, B, and W). The calibration process itself terminates when you have completed the final iteration.

Once calibration (arbitrary calibration or display-characteristics input) has terminated normally, the calibration information is written into the memory channel for the selected probe connector. The written information includes: calibration mode information, the calibration data, the standard (white) data, and the serial number of the calibrated probe.

Standard white setting can be carried out independently of arbitrary calibration or display-characteristics input. To set the white setting independently, carry out the above procedure for the W pattern only. If you are using arbitrary calibration mode, you can also input the numeric settings manually (W values only). The following lists are the sample program for calibration attached in the SDK. (They are some parts of the sample program for reference. So you cannot complete the calibration certainly, although you perform all of them.)

Example: Executing Matrix Calibration and White Calibration

Declaring the variables

```
Public Type TypeReferenceData
    sRefx As Single          ' Calibration data, x
    sRefy As Single          ' Calibration data, y
    sRefLv As Single         ' Calibration data, Lv
End Type

Private typCalData(4) As TypeReferenceData
```

Excuting Calibration

```
    On Error GoTo E

'================================
' Check Cal Data
'================================
bReturn = SetCalData() ' Method to set the calibration data
If bReturn = False Then Exit Sub
```

(Code is omitted.)

```
'================================
' Matrix Calibration
'================================
    If objMemory.ChannelNO = 0 Then
        ' Cannot excute the matrix calibration for Channel 0
       MsgBox "CH00 cannot be calibrated", vbOKOnly
       Exit Sub
    End If

    objCa.DisplayMode = 0                  ' Procedure 2 Lvxy display mode
    objMemory.ChannelNO = ListNo - 1       ' Procedure 3
           ' ListNo is the value of the defined number for calibration memory channel.
    objCa.SetLvxyCalMode                   ' Procedure 4

    '--------------------------------
    ' Red
    '--------------------------------
    Result = MsgBox("Red Measure", vbOKCancel)    ' Procedure 5 Display the RED
                                                    pattern
    If Result = vbCancel Then
       objCa.ResetLvxyCalMode
       Exit Sub
    End If

    objCa.Measure  ' Procedure 6
    objCa.SetLvxyCalData CLR_RED, typCalData(CLR_RED).sRefx, _
typCalData(CLR_RED).sRefy, typCalData(CLR_RED).sRefLv  ' Procedure 7
```

```
        '--------------------------------
        'Green
        '--------------------------------
(Code is omitted.)
        '--------------------------------
        'Blue
        '--------------------------------
(Code is omitted.)
        '--------------------------------
        'White
        '--------------------------------
(Code is omitted.)

      objCa.Enter  ' Procedure 8

(Code is omitted.)
   '================================
   ' White Calibration
   '================================
      If objMemory.ChannelNO = 0 Then
            ' Cannot excute the white calibration for Channel 0
         MsgBox "CH00 cannot be calibrated", vbOKOnly
         Exit Sub
      End If

      objCa.DisplayMode = DSP_LXY          ' Procedure 2
      objMemory.ChannelNO = ListNo - 11    ' Procedure 3
      objCa.SetLvxyCalMode                 ' Procedure 4

      '--------------------------------
      'White
      '--------------------------------
      Result = MsgBox("Measure White", vbOKCancel) ' Procedure 5 Display the
                                                     WHITE pattern
      If Result = vbCancel Then
         objCa.ResetLvxyCalMode
         Exit Sub
      End If
      objCa.Measure    ' Procedure 6
      objCa.SetLvxyCalData CLR_WHITE, typCalData(CLR_WHITE).sRefx, _
typCalData(CLR_WHITE).sRefy, typCalData(CLR_WHITE).sRefLv  ' Procedure 7
      objCa.Enter      ' Procedure 8

(Code is omitted.)
   '================================
   ' White Set (other than CH00)
   '================================
      If objMemory.ChannelNO = 0 Then
            ' Cannot set standard white data by measurement for Channel 0
         MsgBox "CH00 cannot be set", vbOKOnly
         Exit Sub
      End If
      objCa.DisplayMode = DSP_LXY              ' Procedure 2
      objMemory.ChannelNO = ListNo - 1         ' Procedure 3
      Result = MsgBox("White Measure", vbOKCancel) ' Procedure 5 Display the
                                                     WHITE pattern
      If Result = vbCancel Then
         Exit Sub
      End If
      objCa.Measure       ' Procedure 6
      objCa.Enter         ' Procedure 8
```

```
    '================================
    ' White Data Set (CH00)
    '================================
        If objMemory.ChannelNO <> 0 Then
              ' Cannot set standard white data by inputting value for other than Channel 0
            MsgBox "Only For CH00", vbOKOnly
            Exit Sub
        End If
        objCa.SetLvxyCalData CLR_WHITE, typCalData(CLR_WHITE).sRefx, _
  typCalData(CLR_WHITE).sRefy, typCalData(CLR_WHITE).sRefLv ' Procedure 7

    Else

     (Code is omitted.)
        '================================
        ' Channel ID Set
        '================================
              ' strID is the string set for ID
        If strID = "" Then
            objMemory.SetChannelID "CH" + Str$(ListNo - 1)
        Else
            objMemory.SetChannelID strID
        End If

     (Code is omitted.)

E:
    '================================
    ' Error Trap
    '================================
    If bSetMode = True Then
        objCa.ResetLvxyCalMode
    End If

    Dim strERR As String
    Dim iReturn As Integer

    strERR = "Error from " + Err.Source + Chr$(10) + Chr$(13)
    strERR = strERR + Err.Description + Chr$(10) + Chr$(13)
    strERR = strERR + "HRESULT " + CStr(Err.Number - vbObjectError)
    MsgBox strERR, vbOKOnly
```

The SDK also provides the capability of copying calibration data from a file into a memory channel, and from a memory channel into a file. This feature makes it easy to provide applications with the ability to manage calibration data and memory channels. The following samples show how this feature can be used.

Example: Writing Data from CA-200 to File

```
' Specify the channel number of the channel holding the calibration data to be copied.
objCa.Memory.ChannelNO = 1

' Save the calibration data.
For Each objProbe In objCa
   lProbeSNO = objProbe.SerialNO

   ' Generate calibration data filename (strDataFile) from memory channel, ID, and
   ' probe serial number. (Code is omitted.)

   ' Write the calibration data from the memory channel into the file.
   objCa.Memory.CopyToFile objProbe.Number, strDataFile
Next objProbe
```

Example: Writing Data from File to CA-200

```
' Specify the channel number of the channel holding the calibration data to be copied.
objCa.Memory.ChannelNO = 10

'Specify the memory channel ID (by entering it manually or by extracting it from the
'calibration data name). (Code is omitted.)

' Load the calibration data.
For Each objProbe In objCa
   lProbeSNO = objProbe.SerialNO

   ' Generate calibration data filename (strDataFile) from lsProbeSNO,
   ' calibration data name, etc. (Code is omitted.)

   ' Copy the calibration file's calibration data into the memory channel.
   objCa.Memory.CopyFromFile objProbe.Number, strDataFile
Next objProbe
```

3.2.2 Processing that need to be done before an arbitrary calibration channel is used

Rigorous configuration management also requires management of probes and of the calibration data used with those probes. The identity of probes can be checked using their serial numbers. The calibration information handled by the program, meanwhile, provides not only the raw calibration data but also attaches information indicating the calibration type (that is, whether the data is for arbitrary calibration or for analyzer-mode operation) and the identity of the probe to which the data is associated. Accordingly, you can see whether calibration data is being applied correctly by comparing the probe serial number in the calibration information against the probe serial number of the actually connected probe.

The following sample code gets the serial number of the connected probe from the `SerialNO` property of the `Probe` object, and then uses the `Memory` object's `GetMemoryStatus` method to get the calibration information (calibration type and probe serial number).

Example: Checking Consistency of Probe Configuration and Calibration Data

```
'Declare the SDK object variables.
Public objCa200 As Ca200          'Application object
Public objCa As Ca                'CA-200 object
Public objProbe As Probe          'Probe object
Public objMemory As Memory        'Memory object

' Create the application object.
Set objCa200 = New Ca200

'Set the configuration.
objCa200.SetConfiguration 1, "1234", 1 ,38400        ‘ COM1
objCa200.SetConfiguration 2, "1234", 2 ,38400        ‘ COM2
objCa200.SetConfiguration 3, "1234", 3 ,38400        ‘ COM3

'Check the probes.
For Each objCa In objCa200.Cas      'Call each CA-200, in order.

    'Create memory object for the called CA-200.
    Set objMemory = objCa.Memory
    For Each objProbe In objCa.Probes     'Call each probe, in order.

        'Get ID No. and serial No. of the called probe.
        lNumber = objProbe.Number
        lProbeSNO = objProbe.SerialNO

        ' Check probe configuration (confirm that specified probe is connected to
        '                    the correct port of the specified CA).    (Code is omitted.)


        ' For called probe, call each of the utilized memory channels, in order.
        For ch = 10 To 19

            'Set the memory channel.
            objMemory.ChannelNO = ch

          'Get the channel's calibration information.
          'Gotten lCProbeSNO and lRProbeSNO show for which serial number of probe
          '                    the calibration data and the target color data are.
           objMemory.GetMemoryStatus lNumber,lCProbeSNO,lRProbeSNO, _
          lCalMode
```

```
            'Check consistency with calibration data. (Code is omitted.)
            '(Confirm that probe serial number attached to calibration data
            '           matches the serial number of the connected probe, etc.)

        Next ch
    Next objProbe
Next objCa
```

To check whether the calibration data in the CA-200's memory channel is correct, you can use the `Memory` object's `CheckCalData` method, as shown in the sample code below.

Example: Routine to Check Appropriateness of the Calibration Data in the CA-200

```
'Check the calibration data.
For Each objCa In objCa200.Cas              'Call each CA-200, in order.

    'Create memory object for the called CA-200.
    Set objMemory = objCa.Memory
    For Each objProbe In objCa.Probes       'Call each probe, in order.

        'Get ID No. and serial No. of the called probe.
        lNumber = objProbe.Number
        lProbeSNO = objProbe.SerialNO

        ' Call memory channels, in order.
        For ch = 1 To 99

            'Set the memory channel.
            objMemory.ChannelNO = ch

            'Get the memory channel's ID.
            strMemoryID = objMemory.ChannelID

            ' Determine corresponding calibration data fie (filename: strdataFile) based
            '           on probe serial number, memory channel ID. (Code is omitted.)

            ' Check that the calibration data matches the memory-channel data.
            lr = Memory. CheckCalData(lNumber, strDataFile)


        Next ch
    Next objProbe
Next objCa      'Call remaining calibration data to be checked, in order.
```

3.2.3 Zero-Calibration Event Processing

If the probe temperature changes by a certain amount following zero-calibration, the probe must be zero-calibrated once again. In the event that such re-zeroing is required, the device will return a zero-calibration warning code within the status information of the measurement results. This section briefly explains how applications can handle these codes.

If the SDK detects the zero-calibration warning code within the measurement results, it generates a zero-calibration request event. By setting up a Object which receives the event in the application, you can include code to handle the event and automatically execute zero calibration.

From Object of VB can handle the event. Below is a brief overview **Form Object** creation and event process coding.

③ First, declare a `Ca` object variable. Declare it in **Form Module**, using the `WithEvents` keyword in the declaration.

Example: Declaration of `Ca` Object Variable

```
' Declare Ca Object variable.
Dim WithEvents objCa as Ca
```

Now that it has been ready for defining the **Procedure** at the time of receiving the event in **From Module**, selecting `SyncObject` in the **Object** box will enable you select the zero-calibration event procedure (`ExeCalZero`) in the **Procedure** box. You can now program this procedure to zero-calibrate the relevant CA-200, or to display a message on the CA-200, or to take other relevant action.

Example: Handling of the zero-calibration event

```
Private Sub objCaSync_ExeCalZero()
   MsgBox "Cal Zero"
   objCa.CalZero
End Sub
```

Next, of course, you will also need to link the `SyncObject` to the `Ca` object which is the other party of the communication. The following example shows how to link the `SyncObject` to the SDK's `Ca` object within the main routine such as formatting **Form Object**.

Example: Setting a SycnObject

```
Set objCaSync = objCa
```

In the above code, `objCa` is the SDK's `Ca` object for the targeted CA-200 unit.

3.2.4 Obtaining the hardware information of the connected probes

By using IProbeInfo Object you can maintain the information of the connected hardware (the model No., the model name) and obtain the model No. and the model name from a Property in this object. Declaration of the SDK object variables and assignment of the object variables (generation) will be as follows.

’ Declaring the SDK object variables

```
Public objProbeInfo as IProbeInfo
Public sProbeTypeName as String
```

’ Assigning the object variables (generation)

```
Set objProbeInfo = objProbe
```

’ Obtaining the model No. and the model name of the connected probes

```
sProbeTypeName = objProbeInfo.TypeName
lProbeTypeNo = objProbeInfo.TypeNO
```

3.2.5 Errors: Likely Causes and Appropriate Responses

This section explains issues related to error processing by applications using this SDK.

The SDK can return four types of error, as follows.

① Communication error: Error in USB or RS communication between the CA-200 and the host PC

② CA-200 execution error: CA-200 command error

③ SDK execution error: Error caused by operation of SDK method or property

④ COM Error: Error caused by SDK internal object operation

Errors of type ① will generally have one of two causes, as follows.

①-1. Communication-related parameter was improperly set when the configuration was being set up.
①-2. Hardware error (cable connection problem) occurred while the application was running

An error of type ①-2 may cause subsequent problems if you continue operation. If this type of error occurs, the best response is to carry out minimal post-processing and then restart the system.

Note also that while each CA-200 unit provides both a USB port and an RS port, you should never configure them separately and then connect them up at the same time. (The SDK will not check for this, and correct operation cannot be guaranteed.)

Errors of type ④ are execution errors related to the system's COM infrastructure. Errors of this type are not anticipated so long as you are operating on a local server.

Errors of type ② and ③ indicate that execution itself was carried out normally but that the result was abnormal. These errors can be generated by a number of causes, including the following:

(1) Incorrect parameter setting in method or property
(2) Mismatch between method or property and execution environment
(3) Problem with CA-200 operation or measurement environment

Errors of type (1) are produced during application development. For these errors you should refer to the error object (see below) and then take appropriate steps to resolve the problem.

Most errors of type (2) are also produced during application development. These errors occur because some methods and properties cannot be used with certain object states. Refer to the error information and take appropriate steps to resolve the problem.

Some errors of type (2), as well as all errors of type (3), are runtime errors. These errors do not have a significantly adverse impact on SDK operation, and the application can continue running provided that it first clears the error. To clear the error, the application must utilize the error object and implement appropriate error processing.

When an error occurs, the SDK will generate an error object in accordance with the COM error-object protocol. The error object returns the following information:

① Error type (one of the error types indicated above)
② Content of error
③ Hints about how to resolve the error

The sample code shown below implements normal VB error trapping and error processing. The error object returned by the SDK can be referenced using the VB `Err` object, allowing the error to be trapped. Specifically, ID information about the object that returned the error object can be referenced using `Err.Source`; the error information (type, content, and hints) can be referenced using `Err.Description`; and the error type number can be referenced using `Err.Number`.

Example: Error Processing

```
    On Error GoTo Er

    Set objCa200 = New Ca200
    objCa200.AutoConnect
    Set objCa = objCa200.SingleCa
    Set objProbe = objCa.SingleProbe
    Set objMemory = objCa.Memory

    MsgBox "0-Cal", vbOKOnly
    objCa.CalZero
    …
Er:
    Dim strERR As String
    Dim iReturn As Integer

    strERR = "Error from " + Err.Source + Chr$(10) + Chr$(13)
    strERR = strERR + Err.Description + Chr$(10) + Chr$(13)
    strERR = strERR + "HRESULT " + CStr(Err.Number - vbObjectError)
    iReturn = MsgBox(strERR, vbRetryCancel)
    Select Case iReturn
        Case vbRetry: Resume
        Case Else:
            objCa.RemoteMode = 0
            End
End Select
```

4. SDK Reference

				PAGE
Add	Adds a probe to the specified collection of output probes.	OutputProbes Collection	Method	104
AddAll	Adds all connected probes to the collection of output probes.	OutputProbes Collection	Method	105
AutoConnect	Automatically configures a simple CA-200 setup consisting of a single CA-200 unit with a single probe.	Ca200 Object	Method	39
AveragingMode	Sets or gets the CA-200 unit's averaging mode (FAST, SLOW, or AUTO).	Ca Object	Property	58
B	These properties get the analyzer-mode measurement results.	Probe Object	Property	121
BrightnessUnit	Sets or gets the CA-200 unit's brightness display unit.	Ca Object	Property	59
CalStandard	Sets or Gets the CA-200 unit's default calibration mode.(CH00)	Ca Object	Property	66
CalZero	Executes zero-calibration of the CA-200 unit.	Ca Object	Method	67
Cas	Gets collection of connected CA-200 units.	Ca200 Object	Collection	35
CAType	Gets the CA-200 unit's product type.	Ca Object	Property	60
CAVersion	Gets the CA-200 unit's firmware version information.	Ca Object	Property	61
ChannelID	Selects the CA-200 unit's memory channel, or gets the current selection. Selection is expressed by channel ID name.	Memory Object	Property	86
ChannelNO	Specifies the CA-200 unit's memory channel, or gets the current selection. Selection is expressed by channel number.	Memory Object	Property	85
CheckCalData	Compares the calibration data in the currently selected memory channel against calibration data held in the specified calibration data file.	Memory Object	Method	90
Clone	Gets a copy of the collection of output probes.	OutputProbes Collection	Method	107
CopyFromFile	Copies the calibration data from the specified file into the currently selected memory channel.	Memory Object	Method	92
CopyToFile	Copies the calibration data from the currently selected memory channel into a file.	Memory Object	Method	91
Count	Gets the count of the connected CA-200 units.	Cas Collection	Property	43
Count	Gets the count of the connected probes.	Probes Collection	Property	96
Count	Gets the count of the output probes.	OutputProbes Collection	Property	102
DisplayDigits	Sets or gets the number of digits displayed on the CA-200 unit (the CA-200 unit's display-digits setting).	Ca Object	Property	57
DisplayMode	Sets or gets the CA-200 unit's display mode (measuring mode).	Ca Object	Property	56
DisplayProbe	Specifies or gets the display probe of the CA-200 unit.	Ca Object	Property	53
duv	Use these properties to get the correlated color temperature and the difference from black-body locus, as represented in uv color space.	Probe Object	Property	116
Enter	Writes calibration data (arbitrary calibration data, matrix calibration data, or display-characteristics data) into memory.	Ca Object	Method	78
ExeCalZero	Notifies the client that the CA-200 unit requires zero-calibration.	Ca Object	Method	82
FlckrFMA	Gets the FMA flicker measurement.(Only the applicable model)	Probe Object	Property	118
FlckrJEITA	Gets the JEITA flickermeasurement.(Only the applicable model)	Probe Object	Property	117
G	These properties get the analyzer-mode measurement results.	Probe Object	Property	121
GetAnalogRange	Gets the range of the CA-200 unit's analog display.	Ca Object	Method	70
GetFMAAnalogRange	Gets the analog display range used by the CA-210 for flicker measurement.(Only the applicable model)	Ca Object	Method	72
GetMemoryStatus	Gets calibration information from the currently selected memory channel.	Memory Object	Method	89
GetReferenceColor	Gets the reference (white) color setting for the selected memory channel.	Memory Object	Method	87
GetSpectrum	Gets the amplitude of each frequency in the JEITA flicker measuring data.	Probe Object	Property	122
ID	Sets or gets the ID name of the CA-200 unit.	Ca Object	Property	64
ID	Sets or gets the ID name for the targeted probe.	Probe Object	Property	120
Item	Gets the specified CA-200 unit from the collection of connected CA-200 units.	Cas Collection	Property	42
Item	Gets the specified probe from the collection of probes connected to the CA-200 unit.	Probes Collection	Property	95
Item	Gets the specified probe from the collection of output probes.	OutputProbes Collection	Property	101
ItemOfNumber	From the collection of connected CA-200 units, gets the CA-200 unit identified by the specified ID number.	Cas Collection	Property	44
ItemOfNumber	From the collection of connected probes, gets the probe identified by the specified ID number.	Probes Collection	Property	97
ItemOfNumber	From the collection of output probes, gets the probe identified by the specified ID number.	OutputProbes Collection	Property	103
LsUser	These properties get the results in u'v' or L*u*v* color space.	Probe Object	Property	115
Lv	These properties get the brightness measurement results in cd/m2 (Lv) or fL (LvfL) units.	Probe Object	Property	114
LvfL	These properties get the brightness measurement results in cd/m2 (Lv) or fL (LvfL) units.	Probe Object	Property	114

				PAGE
Measure	Executes measurement.	Ca Object	Method	68
Memory	Gets the memory channel space of the corresponding CA-200 unit.	Ca Object	Property	52
Number	Gets the CA-200 unit's ID number.	Ca Object	Property	62
Number	Gets the probe's ID number.	Probe Object	Property	119
OutputProbes	Gets the collection of output probes.	Ca Object	Property	51
PortID	Gets the ID of the CA-200 unit's communication port.	Ca Object	Property	63
Probes	Gets the collection of the probes connected to the specified CA-200 unit.	Ca Object	Property	50
R	These properties get the analyzer-mode measurement results.	Probe Object	Property	121
RAD	Use these properties to get the measurement-results status code for the corresponding measurement (color measurement, JEITA flicker measurement, FMA flicker measurement, and analyzer-mode, respectively).	Probe Object	Property	108
RD	Use these properties to get the measurement-results status code for the corresponding measurement (color measurement, JEITA flicker measurement, FMA flicker measurement, and analyzer-mode, respectively).	Probe Object	Property	108
ReceiveMsr	Receives measurement results from all connected CA-200 units.	Cas Collection	Method	46
RemoteMode	Sets the CA-200 unit's remote mode.(ON/OFF)	Ca Object	Property	65
RemoveAll	Deletes all designature of output probes.	OutputProbes Collection	Method	106
ResetAnalyzerCalMode	Takes the CA-200 unit out of display-characteristics input mode, and returns it to normal mode.	Ca Object	Method	76
ResetLvxyCalMode	Takes the CA-200 unit out of arbitrary calibration mode and returns it to normal mode.	Ca Object	Method	80
RFMA	Use these properties to get the measurement-results status code for the corresponding measurement (color measurement, JEITA flicker measurement, FMA flicker measurement, and analyzer-mode, respectively).	Probe Object	Property	108
RJEITA	Use these properties to get the measurement-results status code for the corresponding measurement (color measurement, JEITA flicker measurement, FMA flicker measurement, and analyzer-mode, respectively).	Probe Object	Property	108
SendMsr	Sends the Measure command to all connected CA-200 units.	Cas Collection	Method	45
SerialNO	Gets the probe's serial number.	Probe Object	Property	119
SetAnalogRange	Sets the range of the CA-200 unit's analog display.	Ca Object	Method	69
SetAnalyzerCalData	Executes input of display-characteristics data (standard values).	Ca Object	Method	77
SetAnalyzerCalMode	Sets the CA-200 unit into display-characteristics input mode.	Ca Object	Method	75
SetCaID	Sets an ID name (alias) for a specified Ca object.	Cas Collection	Method	47
SetChannelID	Sets an ID name for the currently selected memory channel.	Memory Object	Method	88
SetConfiguration	Sets up the CA-200 configuration.	Ca200 Object	Method	37
SetDisplayProbe	Designates the probe that will be used as the CA-200's display probe.	Ca Object	Method	74
SetFMAAnalogRange	Sets the analog display range used by the CA-210 for flicker measurement.(Only the applicable model)	Ca Object	Method	71
SetLvxyCalData	Executes input of arbitrary calibration data (measured values and calibration values).	Ca Object	Method	81
SetLvxyCalMode	Sets the CA-200 unit into arbitrary calibration mode.	Ca Object	Method	79
SetProbeID	Sets an ID name (alias) for a specified Probe object.	Probes Collection	Method	98
SetPWRONStatus	Sets the CA-200 into PWRON state.	Ca Object	Method	73
SingleCa	Gets the CA-200 unit that has been connected by the AutoConnect method.	Ca200 Object	Property	37
SingleProbe	Gets the probe configured by the Ca200 object's AutoConnect method.	Ca Object	Property	54
sx	These properties get the measurement results in xy color space.	Probe Object	Property	113
sy	These properties get the measurement results in xy color space.	Probe Object	Property	113
SyncMode	Sets or gets the CA-200 unit's sync mode.	Ca Object	Property	55
T	Use these properties to get the correlated color temperature and the difference from black-body locus, as represented in uv color space.	Probe Object	Property	116
TypeName	Obtains a character string that indicates the type of the connected probes.	IProbeInfo Object	Property	125
TypeNO	Obtains a value that indicates the type of the connected probe.	IProbeInfo Object	Property	126
ud	These properties get the results in u'v' or L*u*v* color space.	Probe Object	Property	115
usUser	These properties get the results in u'v' or L*u*v* color space.	Probe Object	Property	115
vd	These properties get the results in u'v' or L*u*v* color space.	Probe Object	Property	115
vsUser	These properties get the results in u'v' or L*u*v* color space.	Probe Object	Property	115
X	These properties get the measurement results in XYZ color space.	Probe Object	Property	112
Y	These properties get the measurement results in XYZ color space.	Probe Object	Property	112
Z	These properties get the measurement results in XYZ color space.	Probe Object	Property	112

4.1 Ca200 Object

The `Ca200SK` object is the root object (application object) of the object hierarchy provided by this SDK.

Explanation:

The `Ca200` object is the root object provided by the SDK. This object provides the capability to connect up, configure, and manage up to five CA200 units (below, "CA-200s" or "CA-200 units").

Applications developed with this SDK begin by instantiating this `Ca200` object. The application can then use the `Ca200.SetConfiguration` method (with the relevant arguments) to configure the CA-200s to be used for measurement. The SDK automatically generates the relevant lower-level objects in accordance with these configuration arguments. The application then uses the lower-level objects to control each of the connected CA-200s.

`Ca200` Object: Properties, Methods, and Collections

Properties/ Collections:

`Cas`

`SingleCa`

Methods:

`SetConfiguration`

`AutoConnect`

Events:

None

Ca200::Cas Collection

Gets collection of connected CA-200 units.

Syntax:

```
dim objCas as Cas
Set objCas = objCa200.Cas
```

objCas: Collection of connected CA-200 units (`Cas`)
ObjCa200: Targeted root object provided by the SDK (`Ca200`)

Explanation:

Use the `Cas` property of the `Ca200` object to get the collection of connected CA-200 devices. The `Cas` property is the collection of `Ca` objects representing the currently connected CA-200 units. You can use the `Cas` collection's methods and properties to get the `Ca` object of any of the CA-200 devices in the collection.

Ca200::SingleCa

Gets the CA-200 unit that has been connected by the `Ca200` object's `AutoConnect` method.

Syntax:

```
Dim objCa as Cas
Set objCa = objCa200.SingleCa
```

objCa: CA-200 unit obtained from the `SingleCa` property (`Ca`)
ObjCa200: Targeted root object provided by the SDK (`Ca200`)

Explanation:

If you are using a single USB-connected CA-200 unit with a single probe, you can use the `AutoConnect` method to set up the configuration. (The `AutoConnect` method provides easier setup than the `SetConfiguration` method.) The `SingleCa` property is the `Ca` object that is created by the `AutoConnect` method. You use the methods and properties of this object to control the connected CA-200.

Ca200::SetConfiguration

Sets up the CA-200 configuration.

Syntax:

```
objCa200.SetConfiguration lNumber, strConnectstring, lPort, lBaudrate
```

ObjCa200: Targeted root object provided by the SDK (Ca200)
lNumber: ID number of the CA-200 to be set up (Input: Long)
strConnectstring: Character string indicating probes to be connected (Input: String)
lPort: Type of communication port used for the connection (Input: Long)
lBaudrate: Baud rate for RS communication (Input: Long)

Explanation:

You use the SetConfiguration method to set up the CA-200 configuration.

You must execute this method once for each CA-200 that you want to set up. For arguments, you supply an ID number for the CA-200 to be connected, a character string indicating how probes are to be connected to the CA-200, and a number indicating the type of communications port the CA-200 will be connected to. You must also specify the baud rate at which communication is to be carried out.

When connecting multiple CA units via USB:

0 to 4: USB

It is not possible to mix USB connections with RS COM port connections. Also, the numbers must be in sequence starting with 0.

When using the COM port to connect multiple CA units, the numbers indicating port type are as follows:

lPort=0: USB

lPort=1 to 255: COM1 to COM255

And only 1 unit can be connected via USB (Same specification as previous version 3.12 or before of CA-SDK).

The method checks that the actual hardware is consistent with the supplied arguments, and if there is no inconsistency it automatically generates the corresponding Cas collection, Ca object, Probes collection, OutputProbes collection, and Probe objects. If a contradiction is encountered, the method terminates in error.

The ID number value (*lNumber*), which must be an integer value from 1 to 5, becomes the Ca.Number property of the generated Ca object. The method also automatically generates a corresponding ID name (alias), which is set into the Ca.ID property. (The automatically generated name is "C*a*x", where *x* corresponds to the ID number. For example, if the ID number is 1, then the automatically generated name is "Ca1".) This Ca.ID property can thereafter be used to target specific Ca objects within the Cas collection of Ca objects.

The *strConnectstring* argument gives the probe numbers of the probes to be connected. Probe numbers run from 1 to 5, and the argument is written as a concatenated string. A value of "12345", for example, would connect up all five probes. For each probe, the method generates a corresponding Probe object, and sets the probe number into that object's Probe.Number property. Again, the method also automatically generates a corresponding probe ID name (alias), which is set into the Probe.ID property. (The automatically generated name is "P*x*", where *x* corresponds to the probe number.) The Probe.ID property can thereafter be used to target specific Probe objects within the Probes collection.

Once the `SetConfiguration` method has terminated normally, the application can use the methods and properties of the generated objects to control and manage the connected CA-200 and probes.

Even if using a USB port connection, you must specify a baud rate.

lBaudrate=300, 600, 1200, 2400, 4800, 9600, 19200, 38400

Note:

When performing the `SetConfiguration` method, the CA-SDK determines whether multiple-unit connection is being performed via USB ports or via RS COM ports in the following way:

- If the port number is not 0 and the port number used in the first `SetConfiguration` method is 4 or less, connection via USB is attempted first. If USB connection succeeds, then operation is performed as if multiple units are connected via USB. If connection fails, connection via RS COM is attempted. If connection succeeds, operation is performed as if multiple units are connected via RS COM ports (for port numbers other than 0).
- If the port number is not 0 and the port number used in the first `SetConfiguration` method is 5 or greater, connection via RS COM is attempted. If connection succeeds, operation is performed as if multiple units are connected via RS COM ports (for port numbers other than 0).

Ca200::AutoConnect

Automatically configures a simple CA-200 setup consisting of a single CA-200 unit with a single probe.

Syntax:

```
objCa200.AutoConnect
```

ObjCa200: Targeted root object provided by the SDK (`Ca200`)

Explanation:

The `AutoConnect` method automatically sets the configuration for a single USB-connected CA-200 unit connected to a single probe. The method checks the hardware of the connected CA-200, and if the configuration is normal it proceeds to create the obj`Ca` object and `Probe` object for the connected CA-200 and probe. If the method is unable to set up the configuration, it terminates in error.

To get the `Ca` object and `Probe` object generated by this method, the application uses the `Ca200` object's `SingleCa` and `SingleProbe` properties, respectively.

Once `AutoConnect` has terminated normally, the application can use the methods and properties of the generated objects to control and manage the connected CA-200 and probe.

4.2 Cas Collection

Collection of connected CA-200 units.

Explanation:

When you connect a CA-200 unit using the `Ca200.SetConfiguration` method, the corresponding `Ca` object is added as a member to the `Cas` collection. You use the `Cas` collection's methods and properties to get the `Ca` object of any of the connected CA-200 devices.

`Cas` Collection: Properties, Methods, and Collections

Properties/Collections:

`Item`
`Count`
`ItemOfNumber`

Methods:

`SendMsr`
`ReceiveMsr`
`SetCaID`

Events:

None

Cas::Item

Gets the specified CA-200 unit from the collection of connected CA-200 units.

Syntax:

```
Dim objCa as Ca
Set objCa = objCas.Item(vIndexOrID)
```

objCa: Retrieved CA-200 unit (`Ca`)
ObjCas: Targeted collection of connected CA-200 units (`Cas`)
vIndexOrID: Index value or ID name of the retrieved CA-200 unit (Input: `Variant`)

Explanation:

Use this property to access individual connected CA-200 units. Identify the CA-200 unit either by its index value (1 to 5) or its ID name (“CA1” to “CA5”, or specified ID name).

Cas::Count

Gets the count of the connected CA-200 units.

Syntax:

```
lCount = objCas.Count
```

lCount: Number of connected CA-200 units (Long)
ObjCas: Targeted collection of connected CA-200 units (Cas)

Explanation:

This property gets the count of the connected CA-200 units.

Cas:: ItemOfNumber

From the collection of connected CA-200 units, gets the CA-200 unit identified by the specified ID number.

Syntax:

```
Dim objCa as Cas
Set objCa = objCas.ItemOfNumber(lNumber)
```

objCa: Retrieved CA-200 unit (Output: Ca)
ObjCas: Targeted collection of connected CA-200 units (Cas)
lNumber: ID number identifying a CA-200 unit (Input: Long)

Explanation:

Gets the Ca object corresponding to the ID number given by the *lNumber* argument. Valid range is 1 to 5.

Cas:: SendMsr

Sends the `Measure` command to all connected CA-200 units.

Syntax:

```
objCas.SendMsr
```

`ObjCas`: Targeted collection of connected CA-200 units (`Cas`)

Explanation:

There are two ways to take and return measurements. One way is to use the `Ca` object's `Measure` method. This will send the `Measure` command to the corresponding CA-200 unit, and then receive the results.

The other way is to use the `Cas` collection's `SendMsr` and `ReceiveMsr` methods. The `SendMsr` method sends the `Measure` command to *all* of the connected CA-200 units, and the `ReceiveMsr` method can then be used to receive all of the results. In the case of meny CA-200 units connected, this approach provides better response time than the `Ca` object's `Measure` method.

Note:

Use this method only when the `Measure` method causes any problem. Try the `Measure` method, if even this `SendMsr` method cannot provide the response time.

After completing `SendMsr` method correctly, then it is neccessaly to excute `ReceiveMsr` method. `SendMsr` method performs under this way because it is specified for providing response time. If it isn’t used under this way, some problem on communication with the CA-200 may occur.

Cas:: ReceiveMsr

Receives measurement results from all connected CA-200 units.

Syntax:

```
objCas.ReceiveMsr
```

ObjCas: Targeted collection of connected CA-200 units (`Cas`)

Explanation:

After using the `Cas` collection's `SendMsr` method to command the CA-200 units to take measurement, you can use the `ReceiveMsr` command to receive the measurement results.

While it is is possible to use the `Ca` object's `Measure` method to take measurement and receive results, the use of the `SendMsr` and `ReceiveMsr` pair provides faster response.

Cas:: SetCaID

Sets an ID name (alias) for a specified `Ca` object.

Syntax:

```
objCas.SetCaID lNumber, strID
```

objCas: `Cas` object containing the `Ca` object whose ID is being set
lNumber: ID number identifying a CA-200 unit (Input: `Long`)
strID: ID name for Ca object (Input: `String`)

Explanation:

This method sets an arbitrary ID name that can be used to refer to the specified `Ca` object. While it is always possible to refer to the `Ca` object by its ID number (1 to 5) or by its index value within the `Cas` collection, users may find it more convenient to refer to it by a user-assigned name.

4.3 CA Object

The `Ca` object represents a connected CA-200 unit.

Explanation:

Each `Ca` object represents a physically connected CA-200 unit. You control the CA-200 unit using the properties and methods of its `Ca` object. The `Ca` object supports almost all of the control capabilities supported by the CA-200 unit itself.

Ca Object: Properties, Methods, and Collections

Properties/Collections:

Probes
OutputProbes
Memory
DisplayProbe
SingleProbe
SyncMode
DisplayMode
DisplayDigits
AveragingMode
BrightnessUnit
CaType
CaVersion
Number
PortID
ID
RemoteMode
CalStandard

Methods:

CalZero
Measure
SetAnalogRange
GetAnalogRange
SetFMAAnalogRange
GetFMAAnalogRange
SetPWRONStatus
SetDisplayProbe
SetAnalyzerCalMode
ResetAnalyzerCalMode
SetAnalyzerCalData
Enter
SetLvxyCalMode
ResetLvxyCalMode
SetLvxyCalData

Events:

ExeCalZero

Ca ::Probes

Gets the collection of the probes connected to the specified CA-200 unit.

Syntax:

```
Dim objProbes as Probes
Set objProbes = objCa.Probes
```

objProbes: Retrieved collection of connected probes (`Probes`)
objCa: Targeted CA-200 unit (`Ca`)

Explanation:

When the `Ca200` object's `SetConfiguration` method is used to set up the CA-200 configuration, the method automatically instantiates a `Probe` object for each of the connected probes. It also adds these `Probe` objects to the `Probes` collection. You use the methods and properties of the `Probes` collection to get the `Probe` object for the probe you wish to work on.

Ca ::OutputProbes

Gets the collection of output probes.

Syntax:

```
Dim objOutputProbes as OutputProbes
Set objOutputProbes = objCa.OutputProbes
```

objOutputProbes: Retrieved collection of output probes (`OutputProbes`)
objCa: Targeted CA-200 unit (`Ca`)

Explanation:

The `OutputProbes` collection is the collection of `Probe` objects corresponding to the connected probes that have been designated as output probes.

You use the methods of the `OutputProbes` collection to designate specified connected probes as output probes, or to remove such designation. (When you designated a probe as an output probe, it is added as member to the `OutputProbes` collection.) You use the properties of the `OutputProbes` collection to get specific output probes.

Ca ::Memory

Gets the memory channel space of the corresponding CA-200 unit.

Syntax:

```
Dim objMemory as Memory
Set objMemory = objCa.Memory
```

objMemory: Retrieved memory channels (Memory)
objCa: Targeted CA-200 unit (Ca)

Explanation:

The Ca object's Memory property gets the Memory object for the corresponding CA-200 unit. You use the Memory object's methods and properties to manipulate the memory channels of the CA-200 unit.

Ca ::DisplayProbe

Specifies or gets the display probe of the CA-200 unit.

Syntax:

①Setting

```
objCa.DisplayProbe = strID
```

objCa: Targeted CA-200 unit (Ca)
strID: Probe ID of the probe being set as the display probe (String)

②Obtainment

```
strID = objCA.DisplayProbe
```

objCa: Targeted CA-200 unit (Ca)
strID: Probe ID of the display probe (String)

Explanation:

Use the Ca object's DisplayProbe property to designate one of the CA-200's probes as the display probe, or to get the probe currently designated as the display probe.

Ca::SingleProbe

Gets the probe configured by the `Ca200` object's `AutoConnect` method.

Syntax:

```
Dim objProbe as Probe
Set objProbe = objCa.SingleProbe
```

objProbe: Obtained probe (`Probe`)
objCa: Targeted CA-200 unit (`Ca`)

Explanation:

If you are using a single USB-connected CA-200 unit with a single probe, you can use the `AutoConnect` method to set up the configuration. (The `AutoConnect` method provides easier setup than the `SetConfiguration` method.) The `SingleProbe` property is the `Probe` object that is created by the `AutoConnect` method. You use the methods and properties of this object to get the measurement results from the connected CA-200.

Ca ::SyncMode

Sets or gets the CA-200 unit's sync mode.

Syntax:

①Setting

objCa.SyncMode = *fSyncMode*

objCa: Targeted CA-200 unit (Ca)

fSyncMode: Sync mode setting (Single)

②Obtainment

fSyncMode = *objCa*.SyncMode

fSyncMode: Sync mode setting (Single)

objCa: Targeted CA-200 unit (Ca)

Explanation:

Use this property to get or set the sync mode of the corresponding CA-200 unit. Note that the *fSyncMode* argument is type single. Argument values are as follows.

fSyncMode = 0. :	NTSC
fSyncMode = 1. :	PAL
fSyncMode = 2. :	EXT
fSyncMode = 3. :	UNIV
fSyncMode = 40. to 200. :	INT

Ca ::DisplayMode

Sets or gets the CA-200 unit's display mode.

Syntax:

①Setting

objCa.DisplayMode = *lDisplayMode*

objCa: Targeted CA-200 unit (Ca)

lDisplayMode: Display mode setting (Long)

②Obtainment

lDisplayMode = *objCa*.DisplayMode

lDisplayMode: Display mode setting (Long)

objCa: Targeted CA-200 unit (Ca)

Explanation:

Use this property to get or set the display mode of the corresponding CA-200 unit. Argument values are as follows.

lDisplayMode = 0 : Lvxy
lDisplayMode = 1 : Tdudv
lDisplayMode = 2 : Analyzer mode (no display)
lDisplayMode = 3 : Analyzer mode (G standard)
lDisplayMode = 4 : Analyzer mode (R standard)
lDisplayMode = 5 : u'v'
lDisplayMode = 6 : FMA flicker[*1]
lDisplayMode = 7 : XYZ
lDisplayMode = 8 : JEITA flicker[*2] (No display on CA-200 unit)

[*]1. Contrast flicker method
[*]2. JEITA flicker method

Ca ::DisplayDigits

Sets or gets the number of digits displayed on the CA-200 unit (the CA-200 unit's display-digits setting).

Syntax:

①Setting

```
objCa.DisplayDigits = lDisplayDigits
```

objCa: Targeted CA-200 unit (Ca)

lDisplayDigits: Display-digits setting (Long)

②Obtainment

```
lDisplayDigits = objCa.DisplayDigits
```

lDisplayDigits: Display-digits setting (Long)

objCa: Targeted CA-200 unit (Ca)

Explanation:

Use this property to get or set the display-digits setting of the corresponding CA-200 unit. Argument values are as follows.

lDisplayDigits = 0 : 3-digit display

lDisplayDigits = 1 : 4-digit display

Ca ::AveragingMode

Sets or gets the CA-200 unit's averaging mode (FAST, SLOW, or AUTO).

Syntax:

①Setting

```
objCa.AveragingMode = lAveragingMode
```

objCa: Targeted CA-200 unit (Ca)

lAveragingMode: Averaging-mode setting [FAST, SLOW, or AUTO] (Long)

②Obtainment

```
lAveragingMode = objCa.AveragingMode
```

lAveragingMode: Averaging-mode setting [FAST, SLOW, or AUTO] (Long)

objCa: Targeted CA-200 unit (Ca)

Explanation:

Use this property to get or set the averaging mode of the corresponding CA-200 unit. Argument values are as follows.

lAveragingMode = 0 : SLOW

lAveragingMode = 1 : FAST

lAveragingMode = 2 : AUTO

Ca ::BrightnessUnit

Sets or gets the CA-200 unit's brightness display unit.

Syntax:

①Setting

```
objCa.BrightnessUnit = lBrightnessUnit
```

objCa: Targeted CA-200 unit (Ca)
lBrightnessUnit: Brightness unit setting (Long)

②Obtainment

```
lBrightnessUnit = objCa.BrightnessUnit
```

lBrightnessUnit: Brightness unit setting (Long)
objCa: Targeted CA-200 unit (Ca)

Explanation:

Use this property to get or set the CA-200 unit's brightness display unit. Argument values are as follows.

lBrightnessUnit = 0 : fL
lBrightnessUnit = 1 : cd/m^2

Ca ::CAType

Gets the CA-200 unit's product type.

Syntax:

```
strCAaType = objCa.CAType
```

objCa: Targeted CA-200 unit (Ca)
strCAType: Product type information from targeted CA-200 unit (String)

Explanation:

This property gets the CA-200 unit's product type information.

<Character string of the type>

strCAType ="CA-100Plus"	The CA-200 unit's product type is CA-100Plus.
strCAType ="CA-210"	The CA-200 unit's product type is CA-210.

Property

Ca ::CAVersion

Gets the CA-200 unit's firmware version information.

Syntax:

```
strCAVersion = objCa.CAVersion
```

objCa: Targeted CA-200 unit (`Ca`)
strCAVersion: CA-200 unit's firmware version information (`String`)

Explanation:

This property gets the CA-200 unit's firmware version information.

<Example of character string>

```
"Ver.2.00.0000"
```

Ca ::Number

Gets the CA-200 unit's ID number.

Syntax:

```
lNumber = objCa.Number
```

objCa: Targeted CA-200 unit (Ca)
lNumber: CA-200 unit's ID No. (Long)

Explanation:

This property gets the CA-200 unit's ID number (1 to 5).

Property

Ca ::PortID

Gets the ID of the CA-200 unit's communication port.

Syntax:

```
strPortID = objCa.PortID
```

strPortID: CA-200 unit's comm port ID (String)
objCa: Targeted CA-200 unit (Ca)

Explanation:

This property gets the ID of the CA-200 unit's communication port. Argument values are as follows.

strPortID = "USB" : USB
strPortID = "COM1".. "COM255" : COM1-COM255

Ca :: ID

Sets or gets the ID name of the CA-200 unit.

Syntax:

①Setting

```
objCa.ID = strID
```

objCa: Targeted CA-200 unit (`Ca`)
strID: ID name of the CA-200 unit (`String`)

②Obtainment

```
strID = objCa.ID
```

strID: ID name of the CA-200 unit (`String`)
objCa: Targeted CA-200 unit (`Ca`)

Explanation:

The client can use this property to set an ID name (alias) for the CA-200, or to get the current ID name setting from the CA-200. The ID name can be used to target a specific `Ca` object from the `Cas` collection.

The default settings are “CA1”, “CA2”, … for 1, 2, … of the CA ID number specified by *`SetConfiguration`* Method of `Ca200` Object.

Ca :: RemoteMode

Sets the CA-200 unit's remote mode.

Syntax:

```
objCa.RemoteMode = lRemoteMode
```

objCa: Targeted CA-200 unit (Ca)
lRemoteMode: Remote mode setting for the CA-200 unit (Long)

Explanation:

Use this property to set the CA-200's remote mode to ON, OFF, or LOCKED. If you set the mode to LOCKED, the CA-200 unit will no longer accept input from it's own operating panel. To release the lock, you must reset the remote mode to OFF.

To execute the *SetConfiguration* or *AutoConnect* Method of Ca200 Object sets the mode to LOCKED automaticaly.

Argument values are as follows.

lRemoteMode = 0 : OFF
lRemoteMode = 1 : ON
lRemoteMode = 2 : LOCKED

Ca :: CalStandard

Sets or Gets the CA-200 unit's default calibration mode.

Syntax:

①Setting

```
objCa.CalStandard = lCalStandard
```

lCalStandard: Default calibration mode of the targeted CA-200 unit. (Long)

objCa: Targeted CA-200 unit (Ca)

②Obtainment

```
lCalStandard = objCa.CalStandard
```

objCa: Targeted CA-200 unit (Ca)

lCalStandard: Default calibration mode of the targeted CA-200 unit. (Long)

Explanation:

This property sets the CA-200 unit's default calibration mode, or gets the CA-200 unit's currently set default calibration mode.

The CA-100Plus cannot be set the default calibration mode to 9300K.

lCalStandard = 1 : 6500K

lCalStandard = 2 : 9300K

Ca ::CalZero

Executes zero-calibration of the CA-200 unit.

Syntax:

objCa.CalZero

objCa: CA-200 unit to be zero-calibrated (Ca)

Explanation:

This method executes zero-calibration of the CA-200 unit identified by the Ca object.

Ca ::Measure

Executes measurement.

Syntax:

objCa.Measure

objCa: Targeted CA-200 unit (Ca)

Explanation:

This method causes the targeted CA-200 unit to execute measurement.

Ca ::SetAnalogRange

Sets the range of the CA-200 unit's analog display.

Syntax:

objCa.SetAnalogRange *sRange1*, *sRange2*

objCa: Targeted CA-200 unit (Ca)
sRange1: Range setting 1 (Input: Single)
sRange2: Range setting 2 (Input: Single)

Explanation:

This method sets the display range of the targeted CA-200 unit's analog display. The valid range is as follows:

$$99 \geq sRange1,2 \geq 0.1$$

You can set the range every 1 steps among 10 to 99 and every 0.1 steps among 0.1 to 9.9.

Ca :: GetAnalogRange

Gets the range of the CA-200 unit's analog display.

Syntax:

objCa.GetAnalogRange *sRange1*, *sRange2*

objCa: Targeted CA-200 unit (Ca)
sRange1: Range getting 1 (Output: Single)
sRange2: Range getting 2 (Output: Single)

Explanation:

This method gets the display range of the targeted CA-200 unit's analog display.

Ca ::SetFMAAnalogRange(Only the applicable model)

Sets the analog display range used by the CA-210 for flicker[*1] measurement.

Syntax:

objCa.SetFMAAnalogRange *sRange*

objCa: Targeted CA-210 unit (Ca)
sRange: Range setting 1 (Input: Single)

Explanation:

This method sets the analog display range used by the CA-210 for flicker[*1] measurement. The valid range is as follows:

$$99 \geq sRange \geq 0.1$$

You can set the range every 1 steps among 10 to 99 and every 0.1 steps among 0.1 to 9.9.

[*]1. Contrast flicker method

Ca ::GetFMAAnalogRange(Only the applicable model)

Gets the analog display range used by the CA-210 for flicker measurement.

Syntax:

objCa.GetFMAAnalogRange *sRange*

objCa: Targeted CA-210 unit (Ca)
sRange: Range getting 1 (Output: Single)

Explanation:

This method gets the analog display range used by the CA-210 for flicker measurement.

Ca ::SetPWROnStatus

Sets the CA-200 into PWRON state.

Syntax:

objCa.SetPWROnStatus

objCa: Targeted CA-200 unit (Ca)

Explanation:

This method sets the CA-200 unit into PWRON state. Specifically, it sets the following:

① DisplayMode
② SyncMode
③ Probe.Number
④ Memory.ChannelNO

Ca :: SetDisplayProbe

Designates the probe that will be used as the CA-200's display probe.

Syntax:

`objCa.SetDisplayProbe lDisplayProbe`

`objCa`: Targeted CA-200 unit (`Ca`)
`lDisplayProbe`: ID number of probe to be used as display probe (Input: `Long`)

Explanation:

This method selects the probe that will be used as the display probe. The probe is designated using its ID number.

Ca ::SetAnalyzerCalMode

Sets the CA-200 unit into display-characteristics input mode.

Syntax:

objCa.SetAnalyzerCalMode

objCa: Targeted CA-200 unit (Ca)

Explanation:

This method sets the targeted CA-200 unit into display-characteristics input mode. After setting the CA-200 unit into this mode, you can proceed to use the other methods related to display-characteristics input. This method is available only in the analyzer-mode.

Please refer to section 3.2.1, Managing CA-200 Calibration and CA-200 Calibration Data for the information how to input display-characteristics in the analyzer-mode.

Ca :: ResetAnalyzerCalMode

Takes the CA-200 unit out of display-characteristics input mode, and returns it to normal mode.

Syntax:

```
objCa.ResetAnalyzerCalMode
```

objCa: Targeted CA-200 unit (`Ca`)

Explanation:

This method, when directed at a CA-200 unit that is in display-characteristics input mode, resets that unit into normal mode. All previously entered display-characteristics input is discarded.

Ca ::SetAnalyzerCalData

Executes input of display-characteristics data (standard values).

Syntax:

```
objCa.SetAnalyzerCalData lClr
```

objCa: Targeted CA-200 unit (Ca)
lClr: Color for which display-characteristics input data is to be entered (Input: Long)

Explanation:

This method executes input of display-characteristics standard values at the targeted CA-200 unit. Specifically, the method causes current measurement results to be input as the characteristic values for the designated color. Argument values are as follows.

lClr = 0 : RED
lClr = 1 : GREEN
lClr = 2 : BLUE
lClr = 3 : WHITE

Ca ::Enter

Writes calibration data (matrix calibration data, white calibration data, or display-characteristics data) into memory.

Syntax:

```
objCa.Enter
```

objCa: Targeted CA-200 unit (Ca)

Explanation:

This method causes the targeted CA-200 to write the already input calibration information (matrix calibration data, white calibration data, or display-characteristics data) into its currently selected memory channel. When the method is finished normally, it is returned to the usual mode.

Ca :: SetLvxyCalMode

Sets the CA-200 unit into arbitrary calibration mode.

Syntax:

objCa.SetLvxyCalMode

objCa: Targeted CA-200 unit (Ca)

Explanation:

This method sets the targeted CA-200 unit into arbitrary calibration mode. After setting the CA-200 unit into this mode, you can proceed to use the other methods related to arbitrary calibration. This method is available only in the color/brightness measurement mode.

Please refer to section 3.2.1, Managing CA-200 Calibration and CA-200 Calibration Data for the information how to execute arbitrary calibration in the color/brightness measurement mode.

Ca :: ResetLvxyCalMode

Takes the CA-200 unit out of arbitrary calibration mode and returns it to normal mode.

Syntax:

objCa.ResetLvxyCalMode

objCa: Targeted CA-200 unit (Ca)

Explanation:

This method, when directed at a CA-200 unit that is in arbitrary calibration mode, resets that unit into normal mode. All previously entered arbitrary-calibration input is discarded.

Ca:: SetLvxyCalData

Executes input of arbitrary calibration data (measured values and calibration values).

Syntax:

```
objCa.SetLvxyCalData lClr, sx, sy, sLv
```

objCa: Targeted CA-200 unit (Ca)
lClr: Color for which data is to be entered (Input: Long)
sx: x calibration value (Input: Single)
sy: y calibration value (Input: Single)
sLv: Lv calibration value (Input: Single)

Explanation:

This method executes input of display-characteristics arbitrary calibration data (measured values and calibration values) at the targeted CA-200 unit. Specifically, the method causes the current measurement results and the argument-passed calibration values to be entered as arbitrary calibration data for the argument-specified color.

lClr = 0 : RED
lClr = 1 : GREEN
lClr = 2 : BLUE
lClr = 3 : WHITE

Ca::ExeCalZero

Notifies the client that the CA-200 unit requires zero-calibration.

Syntax:

```
Dim WithEvents objCaSync As Ca

Set objCaSync = objCa

Private Sub objCaSync_ExeCalZero()
    MsgBox "Cal Zero"
    objca.CalZero
End Sub
```

objCa: Targeted CA-200 unit (`Ca`)

Explanation:

If temperature changes some predetermined amount from what it was at the time of the last zero-calibration, the CA-200 unit will append to its measurement results a message indicating that zero-calibration must be carried out again. Upon detecting this message, the `Ca` object generates the `ExeCalZero` event.

Note that the CA-200 unit itself will display "E2".

4.4 Memory Object

The `Memory` object represents the CA-200 unit's memory channels.

Explanation:

The `Memory` object represents the CA-200 unit's memory channels. You use this object's properties and methods to select and operate on the CA-200 unit's memory channels.

`Memory` Object: Properties, Methods, and Collections

Properties/Collections:

`ChannelNO`
`ChannelID`

Methods:

`GetReferenceColor`
`SetChannelID`
`GetMemoryStatus`
`CheckCalData`
`CopyToFile`
`CopyFromFile`

Events:

None

Memory::ChannelNO

Specifies the CA-200 unit's memory channel, or gets the current selection. Selection is expressed by channel number.

Syntax:

①Setting

```
objMemory.ChannelNO = lCh
```

objMemory: Targeted CA-200's memory channels (Memory)
lCh: Memory channel selection (Long)

②Obtainment

```
lCh = objMemory.ChannelNO
```

lCh: Currently selected memory channel (Long)
objMemory: Targeted CA-200's memory channel space (Memory)

Explanation:

This property selects a memory channel for the CA-200 unit, or returns the current selection. Note that the property's channel argument identifies the channel by its channel number on the CA-200 unit.

Range setting：0～99ch

Memory::ChannelID

Selects the CA-200 unit's memory channel, or gets the current selection. Selection is expressed by channel ID name.

Syntax:

①Setting

```
objMemory.ChannelID = strID
```

objMemory: Targeted CA-200's memory channels (Memory)
strID: Memory channel selection (String)

②Obtainment

```
strID = objMemory.ChannelNO
```

strID: Currently selected memory channel (String)
objMemory: Targeted CA-200's memory channels (Memory)

Explanation:

This property selects a memory channel on the CA-200 unit, or returns the current selection. Note that the property's channel argument identifies the channel by its channel ID name.

Memory::GetReferenceColor

Gets the reference (white) color setting for the selected memory channel.

Syntax:

objMemory.GetReferenceColor *strID*, *sRclr1*, *sRclr2*, *sRclr3*

objMemory: Memory channels (Memory)
strID: ID name of targeted probe (Input: String)
sRclr1: Returned reference (white) color's x value (Output: Single)
sRclr2: Returned reference (white) color's y value (Output: Single)
sRclr3: Returned reference (white) color's Lv value (Output: Single)

Explanation:

This method gets the reference (white) color setting of the CA-200's currently selected memory channel. Note that the channel is identified by its channel ID name (probe ID name).

Memory::SetChannelID

Sets an ID name for the currently selected memory channel.

Syntax:

objMemory.SetChannelID *strID*

objMemory: Targeted memory channel (Memory)
strID: ID name setting for the currently selected memory channel (Input: String)

Explanation:

This method sets an ID name for the currently selected memory channel.

Memory Object

Method

Memory:: GetMemoryStatus

Gets calibration information from the currently selected memory channel.

Syntax:

```
objMemory.GetMemoryStatus lNumber, lCProbeSNO, lRProbeSNO, lCalMode
```

objMemory: Targeted memory channel (Memory)
lNumber: Targeted probe's ID number (Input: Long)
lCProbeSNO: Calibration probe's serial number (Output: Long)
lRProbeSNO: Reference-color probe's serial number (Output: Long)
lCalMode: Calibration mode (Output: Long)

Explanation:

This method gets calibration information about the currently selected memory channel. Specifically, the method returns calibration information that is stored within the targeted probe's calibration data.

<Information of Calibration Mode>

Calibration mode that will be obtained by the ICal Mode is as follows.

	1～99ch		0ch	
CA-210 Universal Measuring Probe (CA-PU12/15)	５０	５１	１	２
CA-210 Small Universal Measuring Probe (CA-PSU12/15)	６０	６１	１	２
CA-210 LCD Flicker Measuring Probe (CA-P12/15)	１０	１１	１	２
CA-210 Small LCD Flicker Measuring Probe (CA-PS12/15)	２０	２１	１	２
CA-100Plus Measuring Probe (CA-P02/05)	１０	１１	１	－
CA-100Plus High Luminance Measuring Probe (CA-PH02/05)	２０	２１	１	－
	White Calibration	Matrix Calibration	Konica Minolta 6500K Matrix Calibration	Konica Minolta 9300K Matrix Calibration

Memory:: CheckCalData

Compares the calibration data in the currently selected memory channel against calibration data held in the specified calibration data file, which was made by `CopyToFile` method of `Memory` object, with ful-path.

Syntax:

```
lResult = objMemory.CheckCalData (lNumber, strFileName)
```

lResult: Result of comparison (Long)
objMemory: Targeted memory channel (Memory)
lNumber: Probe ID number of probe to be checked (Input: Long)
strFileName: Name of calibration data file to be used for comparison (Input: String)
lRProbeSNO: Reference-color probe's serial number (Output: Long)
lCalMode: Calibration mode (Output: Long)

Explanation:

This method compares the calibration data in the currently selected memory channel (the calibration data for the specified probe) against the calibration data stored in the designated calibration data file. If the data match, the method returns a 0; if the data do no match, it returns a nonzero value.

Memory:: CopyToFile

Copies the calibration data from the currently selected memory channel into a file with ful-path.

Syntax:

```
objMemory.CopytoFile lNumber, strFileName
```

objMemory: Targeted memory channel (`Memory`)
lNumber: Probe ID number of probe whose data is to be copied (Input: `Long`)
strFileName: Filename for destination calibration file (Input: `String`)

Explanation:

This method copies data from the currently selected memory channel (the calibration data for the specified probe) into a file with the specified file name.

Memory:: CopyFromFile

Copies the calibration data from the specified file with ful-path into the currently selected memory channel.

Syntax:

```
objMemory.CopyFromFile lNumber, strFileName
```

objMemory: Targeted memory channel (`Memory`)
lNumber: Probe ID number of destination probe (Input: `Long`)
strFileName: Filename of source calibration file (Input: `String`)

Explanation:

This method copies calibration data from the specified file into the designated probe of the currently selected memory channel.

4.5 Probes Collection

Collection of measurement probes connected to the CA-200 unit.

Explanation:

The `Probes` collection is the collection of probes connected to the CA-200 unit.

When you connect a CA-200 unit using the `Ca200.SetConfiguration` method, the corresponding `Probe` objects are created (one for each of the connected probes) and are added as members to the `Probes` collection. You can then use the collection's methods and properties to get the `Probe` object of any of the connected probes.

`Probes` Collection: Properties, Methods, and Collections

Properties/Collections:

`Item`
`Count`
`ItemOfNumber`

Methods:

`SetProbeID`

Events:

None

Probes::Item

Gets the specified probe from the collection of probes connected to the CA-200 unit.

Syntax:

```
Dim objProbe as Probe
Set objProbe = objProbes.Item(vIndexOrID)
```

objProbe: Retrieved probe (`Probe`)
obj.Probes: Collection of connected probes (`Probes`)
vIndexOrID: Index value or ID name of the retrieved probe (Input: `Variant`)

Explanation:

Use this property to get a probe from the collection of probes connected to the CA-200 unit. Identify the probe either by its index value or its ID name.

Property

Probes::Count

Gets the count of the connected probes.

Syntax:

```
lCount = objProbes.Count
```

lCount: Number of connected probes (Long)
objProbes: Collection of connected probes (Probes)

Explanation:

This property gets the count of probes connected to the CA-200 unit. Specifically, it returns the number of Probe objects in the Probes collection.

Probes:: ItemOfNumber

From the collection of connected probes, gets the probe identified by the specified ID number.

Syntax:

```
Dim objProbe as Probe
Set objProbe = objProbes.ItemOfNumber(lNumber)
```

objProbe: Retrieved probe (Probe)
obj.Probes: Collection of connected probes (Probes)
lNumber: ID number identifying the probe (Long)

Explanation:

Gets the Probe object corresponding to the ID number specified by the *lNumber* argument.

Probes::SetProbeID

Sets an ID name (alias) for a specified `Probe` object.

Syntax:

```
objProbes.SetProbeID lNumber, strID
```

objProbes: `Probes` collection containing the `Probe` object whose ID is being set
lNumber: ID number identifying the Probe object whose ID is being set (`Long`)
strID: ID name for Probe object (`String`)

Explanation:

This method sets an arbitrary ID name that can be used to refer to the specified Probe object. While it is always possible to refer to the Probe object by its ID number or by its index value within the Probes collection, users may find it more convenient to refer to it by a user-assigned name.

4.6 OutputProbes Collection

Indicates a collection of output probes.

Explanation：

The `OutputProbes` collection is a collection of `Probe` objects, which are compliant with the probes that are set as output probe.

A connected probe will be set as an Output Probe by method of the `OutputProbes` collection or `SetOutputProbe` method of the `Ca` object. After being set as an output probe, compliant `Probe` objects will be added to the members of the `OutputProbes` collection. Specific Output Probes can be obtained by using method/property of the `OutputProbes` collection.

`OutputProbes` Collection: Properties, Methods, and Collections

Properties/Collections:

`Item`
`Count`
`ItemOfNumber`

Methods:

`Add`
`AddAll`
`RemoveAll`
`Clone`

Events:

None

OutputProbes::Item

Gets the specified probe from the collection of output probes.

Syntax:

```
Dim objProbe as Probe
Set objProbe = objOutputProbes.Item(vIndexOrID)
```

objProbe: Retrieved probe (`Probe`)
objOutoutProbes: Collection of output probes (`OutputProbes`)
vIndexOrID: Index value or ID name of the returned probe (Input: `Variant`)

Explanation:

Use this property to get a specified probe from a collection of output probes.

Property

OutputProbes::Count

Gets the count of the output probes.

Syntax:

```
lCount = objOutputProbes.Count
```

lCount: Number of output probes (Long)
objOutputProbes: Targeted collection of output probes (OutputProbes)

Explanation:

This property gets the count of the probes in the targeted collection of output probes.

OutputProbes::ItemOfNumber

From the collection of output probes, gets the probe identified by the specified ID number.

Syntax:

```
Dim objProbe as Probe
Set objProbe = objOutputProbes.ItemOfNumber(lNumber)
```

objProbe: Retrieved probe (Probe)
objOutputProbes: Targeted collection of output probes (OutputProbes)
lNumber: ID number identifying the output probe to be retrieved (Long)

Explanation:

This property gets, from the specified output probe collection, the output probe corresponding to the ID number given by the *lNumber* argument.

OutputProbes::Add

Adds a probe to the specified collection of output probes.

Syntax:

`objOutputProbes.Add(vIndexOrID)`

`objOutputProbes`: Targeted collection of output probes (`OutputProbes`)
`vIndexOrID`: ID name of the probe to be added to the collection (Input: `String`)

Explanation:

Use this method to designate a probe as an output probe. The method adds the specified probe to the specified collection of output probes.

OutputProbes::AddAll

Adds all connected probes to the collection of output probes.

Syntax:

objOutputProbes.AddAll

objOutputProbes: Targeted collection of output probes (OutputProbes)

Explanation:

Use this method to designate all connected probes as output probes. This method adds all connected probes to the collection of output probes.

OutputProbes:: RemoveAll

Deletes all designature of output probes.

Syntax:

objOutputProbes.RemoveAll

objOutputProbes: Targeted collection of output probes (OutputProbes)

Explanation:

This method deletes the designation of the output probes collection.

OutputProbes:: Clone

Gets a copy of the collection of output probes.

Syntax:

```
Set objProbes = objOutputProbes.Clone
```

objProbes: Copy of the output probes collection (Probes)
objOutputProbes: Targeted collection of output probes (OutputProbes)

Explanation:

This method generates a copy of the output probes collection.

4.7 Probe Object

The `Probe` object represents a probe connected to the CA-200 unit.

Explanation:

Each `Probe` object represents a physically connected probe. When you connect a CA-200 unit using the `Ca200.SetConfiguration` method, the method creates a `Probe` object for each of the probes that are connected to the CA-200. When you execute measurement with a CA-200 unit, the measurement results are reflected in the corresponding `Probe` objects of the CA-200 unit's `Ca` object.

`Probe` Object: Properties, Methods, and Collections

Properties/Collections:

```
RD/RJEITA/RFMA/RAD
X/Y/Z
sx/sy
Lv/LvfL
ud/vd/LsUser/usUser/vsUser/dEUser
T/duv
FlckrJEITA
FlckrFMA
Number/SerialNO
ID
R/G/B
```

Methods:

```
GetSpectrum
```

Events:

None

Probe::RD/RJEITA/RFMA/RAD

Use these properties to get the measurement-results status code for the corresponding measurement (color measurement, JEITA flicker measurement[*1], FMA flicker measurement[*2], and analyzer-mode, respectively).

Syntax:

```
lRD = objProbe.RD
```

lRD: Status code associated with CA's color measurement results (Long)
objProbe: Probe for which code is returned (Probe)

Explanation:

Use these properties to get the status codes for the corresponding CA measurement results.

*1. JEITA flicker method
*2. Contrast flicker method

【Supplement】 About the Measurement Result Property

Even when measurement results are obtained without dysfunctions, you can find out problematic points of the measurement by the return value of Measurement Result Property. Also when measurement value is not normal or error cord is returned, you can find out the cause by Measurement Result Property (below) and Measurement Value Property (the next page).

Example 1 : In X/Y/Z measurement...

Return value of Measurement Value Property（X/Y/Z） : Measurement value (normal)

Return value of Measurement Result Property（RD） : 4

These indicate that the value is outside the measurement range (which corresponds to the flashing indicator on hardware).

Example 2 : In the case of error code 426...

As the cause indicates a wide range, you can find out if the cause is in hardware side or in display setting by using Measurement Result Property and Measurement Value Property, although 'the cause' is defined as 'Either of the probes failed in measurement and that occurred when the display setting was outside the measurement range.

About the Measurement Result Property （RD/RAD/RJEITA/RFMA）

0	Normal completion
1	① A probe in arbitrary calibration/standard color setting is different from a probe in measurement.
2	② An ambient temperature changed by a certain value since a zero calibration.
4	③ Same as outside the measurement range (same as the flashing indicator).
3	④ The above ① and ② occurred at the same time.
5	⑤ The above ① and ③ occurred at the same time.
6	⑥ The above ② and ③ occurred at the same time.
7	⑦ The above ① ② ③ occurred at the same time.
10	⑧ Measurement was executed before zero calibration.
15	⑨ Occurrence of hold error
20	⑩ An invalid external synchronization signal (set an external synchronization signal when less than 40 Hz, over 200 Hz)
22	⑪ Over the measurement range
23	⑫ Offset error
50	⑬ Measurement value is over 100% in the flicker mode.
51	⑭ External synchronizing signal is over 130 Hz in the FMA flicker mode.
52	⑮ Measurement value in the flicker mode is equivalent to Konica Minolta standard white calibration and less than around 0.1cd/m^2 or an equivalent.
53	⑯ Flicker measurement is being attempted using a Universal Measuring Probe CA-PU12 or CA-PU15 or a Small Universal Measuring Probe CA-PSU12 or CA-PSU15.

✱0.1cd/m^2 when the probe, CA-P12/15 is attached. Less than 0.3cd/m^2 or an equivalent when the probe, CA-PS12/15 is attached.

Probe Object

Property

About Measurement Value Property

Measurement value property	Before measurement	After the measurement (normal)	
		Under normal conditions	Under abnormal conditions
X/Y/Z	-1	Measurement value	-1
Lv/LvfL	-1	Measurement value	-1
LsUser/usUser/vsUser/dEUser	-999	Measurement value	-999
FlckrJEITA	-999	Measurement value	-999
R/G/B	-1	Measurement value	-1
sx/sy	-1	Measurement value	-1
ud/vd	-1	Measurement value	-1
T/duv	-1	Measurement value	-1
FlckrFMA	0	Measurement value	-1

When Measurement value property is obtained before the measurement, the property value will be either －1 or －999 (Please see the table above). Also when measurement value property is obtained after the measurement, the measurement value can be obtained under normal conditions but under abnormal conditions it will be －1 or －999. Furthermore, the following cases are possible under abnormal conditions.

	Color 1	Color 2	FMA	JEITA
① Outside the measurement range (over the high intensity)	-1	-1(-999)	-1	-999
② Outside the measurement range (under the low intensity)	Non-applicable	Non-applicable	-1	-999
③ Frequency range is not set as 40～200 Hz	-1	-1(-999)	-1	Non-applicable
④ Frequency range is not set as 40～130 Hz	Non-applicable	Non-applicable	-1	Non-applicable
⑤ Outside the coverage of the microprocessing	Non-applicable	-1(-999)	-1	-999
⑥ Property was obtained before the measurement	-1	-1(-999)	-1	-999

Color 1 : T, duv,usUser,LsUser,dEUser
Color 2: X/Y/Z,Lv,LvfL,sx/sy,R/G/B

Probe::X/Y/Z

These properties get the measurement results in **XYZ** color space.

Syntax:

```
sX = objProbe.X
sY = objProbe.Y
sZ = objProbe.Z
```

sX, *sY*, *sZ*: X, Y, or Z value of measurement result in **XYZ** color space (`Single`)
objProbe: Probe for which result is returned (`Probe`)

Explanation:

Use these properties to get the measurement results as expressed in **XYZ** color space.

Probe::sx/sy

These properties get the measurement results in **xy** color space.

Syntax:

```
sx = objProbe.sx
sy = objProbe.xy
```

sx, *sy*: x or y value of measurement result in **xy** color space (`Single`)
objProbe: Probe for which result is returned (`Probe`)

Explanation:

Use these properties to get the measurement results as expressed in **xy** color space.

Probe::Lv/ LvfL

These properties get the brightness measurement results in cd/m^2 (`Lv`) or fL (`LvfL`) units.

Syntax:

```
sLv = objProbe.Lv
```

sLv: Lv = Y brightness measurement result (`Single`)
objProbe: Probe for which result is returned (`Probe`)

Explanation:

Use these properties to get the brightness measurement results.

Probe:: ud/vd/ LsUser/usUser/vsUser/dEUser

These properties get the results in **u'v'** or **L*u*v*** color space.

Syntax:

```
sfLs = objProbe.LsUser
sus = objProbe.usUser
svs = objProbe.vsUser
```

sLs, *sus*, *svs*: L*, u* and s* measurement results. (Result in **L*u*v*** color space.) (`Single`)
objProbe: Probe for which result is returned (`Probe`)

Explanation:

Use the `Probe` object's `LsUser`, `usUser`, `vsUser`, and `dEUser` properties to get measurement results in **L*u*v*** color space. (These values are calculated with reference to standard white). To get results in **u'v'** color space, use the `Probe` object's `ud` and `vd` properties.

Probe:: T/ duv

Use these properties to get the correlated color temperature and the difference from black-body locus, as represented in **uv** color space.

Syntax:

```
lT = objProbe.T
sduv = objProbe.dUv
```

lT: Correlated color temperature, in **uv** color spaced (`Long`)
sduv: Difference fro black-body locus, in **uv** color space (`Single`)
objProbe: Probe for which result is returned (`Probe`)

Explanation:

Use these properties to get the correlated color temperature and difference from black-body locus, as expressed in **uv** color space.

Probe::FlckrJEITA(Only the applicable model)
Gets the JEITA flicker measurement[*1].

Syntax:

```
sFlckrJEITA = objProbe.FlckrJEITA
```

sFlckrJEITA: Flicker amount (by JEITA measurement[*1]) (`Single`)
objProbe: Probe for which result is returned (`Probe`)

Explanation:

This property gets the JEITA flicker measurement[*1] result.

[*]1. JEITA flicker method

Probe::FlckrFMA(Only the applicable model)
Gets the FMA flicker measurement[*1].

Syntax:

```
sFlckrFMA = objProbe.FlckrFMA
```

sFlckrFMA: Flicker amount (by FMA measurement[*1]) (`Single`)
objProbe: Probe for which result is returned (`Probe`)

Explanation:

This property gets the FMA (AC/DC) flicker measurement[*1] result.

[*1] Contrast flicker method

Probe::Number/SerialNO

Gets the probe's ID number/SerialNO.

Syntax:

```
lNumber = objProbe.Number
lProbeSNO = objProbe.SerialNO
```

objProbe: Targeted probe (`Probe`)
lNumber: Targeted probe's ID number (`Long`)
lProbeSNO: Targeted probe's Serial number (`String`)

Explanation:

This `Number` property returns the probe's ID number.

This `SerialNO` property returns the probe's serial number.

Probe:: ID

Sets or gets the ID name for the targeted probe.

Syntax:

①Setting

objProbe.ID = *strID*

objProbe: Targeted probe (Probe)
strID: ID name for the targeted probe (String)

②Obtainment

```
strID = objProbe.ID
```

strID: Targeted probe's ID name (String)
objProbe: Targeted probe (Probe)

Explanation:

The client can use this property to set an ID name (alias) for the probe, or to get the current ID name setting from the probe. The ID name can be used to target a specific Probe object from the Probes collection.

The default settings are “P1”, “P2”, … for 1, 2, … of the probe ID number specified by *SetConfiguration* method of Ca200 Object.

Probe::R/G/B

These properties get the analyzer-mode measurement results.

Syntax:

```
sR = objProbe.R
sG = objProbe.G
sB = objProbe.B
```

sR, *sG*, *sB*: Red, green, blue measurement result (`Single`)
objProbe: Probe for which result is returned (`Probe`)

Explanation:

Use these properties to return the analyzer-mode measurement results (red, green, and blue results).

Probe::GetSpectrum

Gets the amplitude of each frequency in the JEITA flicker measuring data.

Syntax:

```
sFlckrElement = objProbe.GetSpectrum (lFrequency)
```

sFlckrElement: Amplitude of each frequency (Single)
objProbe: Probe for which result is returned (Probe)
lFrequency: Frequency (Long)

Explanation:

Use the method to get the amplitude of each flicker frequency

$65 \geq$ lFrequency ≥ 6

4.8 IProbeInfo Object

Indicates hardware information of the probes connected to CA-200.

Explanation :

`IProbeInfo` object maintains hardware information (model No./model name) of the connected probes. Model No./model name can be obtained by using properties of the `IProbeInfo` object.

`IProbeInfo` Object

Properties:

`TypeName`

`TypeNO`

Method:

None

Event:

None

IProbeInfo:: TypeName

Obtains a character string that indicates the type of the connected probes.

Syntax :

```
Dim objProbeInfo as IProbeInfo
Dim sProbeTypeName as String

Set objProbeInfo = objProbe
sProbeTypeName = objProbeInfo.TypeName
```

objProbeInfo: Hardware information of the connected probes
sProbeTypeName: A character string of the type to be obtained
objProbe: A connected probe that is an object of the obtainment (Probe)

Explanation :

TypeName property of the IProbeInfo object will be used to obtain a character string that indicates the type of the connected probe.

The character string will be the value below according to the type of the probe.

<Character string of the type>

sProbeTypeName =	"CA-100Plus"	Measuring Probe (CA-P02/05)
	"CA-100PlusH"	High luminance Measuring Probe (CA-PH02/05)
	"CA-210U"	Universal Measuring Probe (CA-PU12/15)
	"CA-210SU"	Small Universal Measuring Probe (CA-PSU12/15)
	"CA-210"	LCD Flicker Measuring Probe (CA-P12/15)
	"CA-210S"	Small LCD Flicker Measuring Probe (CA-PS12/15)

IProbeInfo:: TypeNO

Obtains a value that indicates the type of the connected probe.

Syntax :

```
Dim objProbeInfo as IProbeInfo
Dim lProbeTypeNo as Long

Set objProbeInfo = objProbe
lProbeTypeNo = objProbeInfo.TypeNO
```

objProbeInfo: Hardware information of the connected probe
lProbeTypeNo : Value of the type to be obtained
objProbe : A connected probe that is the object of the obtainment (Probe)

Explanation :

TypeNO property of the IProbeInfo object is used to obtain a value that indicates the type of the connected probe.

Value will be the value below according to the type of the probe.

<Value of the type>

lProbeTypeNo =	1001	: CA-100Plus	Measuring Probe (CA-P02/05)
	1002	: CA-100Plus	High luminance Measuring Probe (CA-PH02/05)
	2102	: CA-210	Universal Measuring Probe (CA-PU12/15)
	2103	: CA-210	Small Universal Measuring Probe (CA-PSU12/15)
	2100	: CA-210	LCD Flicker Measuring Probe (CA-P12/15)
	2101	: CA-210	Small LCD Flicker Measuring Probe (CA-PS12/15)

5. Error Codes

Error No.*	Message (SDK Command Errors)	Cause
401	--cannot execute now --check context	Inappropriate command usage.
402	--invalid argument --check value	Invalid argument.
403	--duplication of name or ID, etc --check context	Value or specification (ID, etc) already exists.
405	--API Fail --Restart Program	SDK-internal API execution error
406	--cannot open cal_data_file --check filename	Failed to open calibration file.
407	--invalid CA number --check value	Invalid CA ID number.
408	--invalid RS/USB ID --check value	Invalid comm-port specification.
409	--invalid Baudrate --check value	Invalid baud rate specification.
410	--null pointer --check program	Invalid pointer argument.
411	--Probe1 should be connected --check argument	Probe #1 not connected.
412	--probe not connected --check program	Designated probe not connected.
413	--invalid x/y --check value	Invalid **x** or **y** value.
414	--invalid Lv --check value	Invalid **Lv** value.
415	--invalid clr --check value	Invalid color specification.
416	--invalid index --check value	Invalid index-value specification.
417	--invalid CA ID --check ID	Invalid CAID specification.
418	--invalid Memory Channel number --check value	Invalid specification of memory-channel number.
419	--invalid Memory Channel ID --check value	Invalid specification of memory-channel ID name.
420	--ID is too long(>10) --check value	Character string for memory-channel ID is too long.
421	--invalid Probe number --check value	Invalid specification of probe No.
422	--invalid Probe ID --check value	Invalid specification of probe ID name.
423	--Data value is too low --check value	Data value is too low.
424	--Data value is too high --check value	Data value is too high.
425	--nonexistent object --check program	Passed a reference to a nonexistent object.
426	--measurement fail --check probe/display_setting	One or more output probes failed to execute measurement. External sync signal not connected.

Error No.*	Message (SDK Command Errors)	Cause
428	--cannot cal channel 0 --check program	Cannot calibrate memory channel 0
429	--No output probes have been specified --check program	No output probes have been specified.
503	--unacceptable calibration data --check value	Invalid specification of calibration value.
504	--unacceptable analog range data --check value	Invalid specification of analog-display range.
506	--matrix calibration error --check calibration data	Error of matrix calibration data. (Invalid calibration data.)
510	--invalid command --check program	Invalid command specification.
520	--no sync_signal --input an external synchronization signal	External sync signal not connected.
521	--too bright --block light	Not dark enough for zero-calibration.
522	--over --set a color within CA's measuring range	Measurement out of range.
523	--offset error --perform zero_calibration	Zero-calibration required.
524	--over in TduvLv mode --set a color within CA's measuring range	Measurement out of range (T⊿uv mode).
553	--flicker error --check probe type	This type of probe cannot measere the flicker.

*The error number is value of the low-order 16 bits of the code returned by the error object.

6. Installing the USB Driver

When you connect to the USB port, the system will automatically prompt for the location of the USB driver. Respond to the prompt by designating the CD-ROM drive and the folder of the OS ('Win2000' for Windows 98, Windows Me, or Windows 2000, 'WinXP' for Windows XP or later).

For installing the USB driver under Windows Vista, it will be automatically installed, when the CD-ROM is set.

For installing the USB driver under Windows XP, the dialog box which confirms User Account Control appears after designating the location of the USB driver. Select 'Continue'.

The program will then install the driver. The driver makes it possible for the CA-200 units to be recognized as USB devices.

9222-1891-16

AHHAPX
Printed in Japan