KEYENCE

LJ-X8000A providers

User's Guide

Version 1.0.0

July 16, 2020

NOTE:

[Revision History]

Version	Date	Description
1.0.0	2020-07-16	First edition

[Operation check model]

Model	Version	Notes
LJ-X8080A		

Contents

1. Introduction

This manual is a user's guide for providers that acquire profile data for LJ-X8000A series of KEYENCE Corporation. Fig. 1-1 shows the overall configuration of this provider and the device. The providers are referred to as LJX8000A providers.

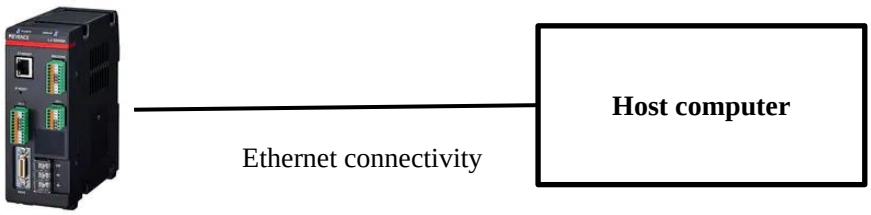


Fig. 1-1 Configuration Diagram

Fig. 1-2 shows the correspondence between this provider and each device.
(※An example. It does not represent everything.)

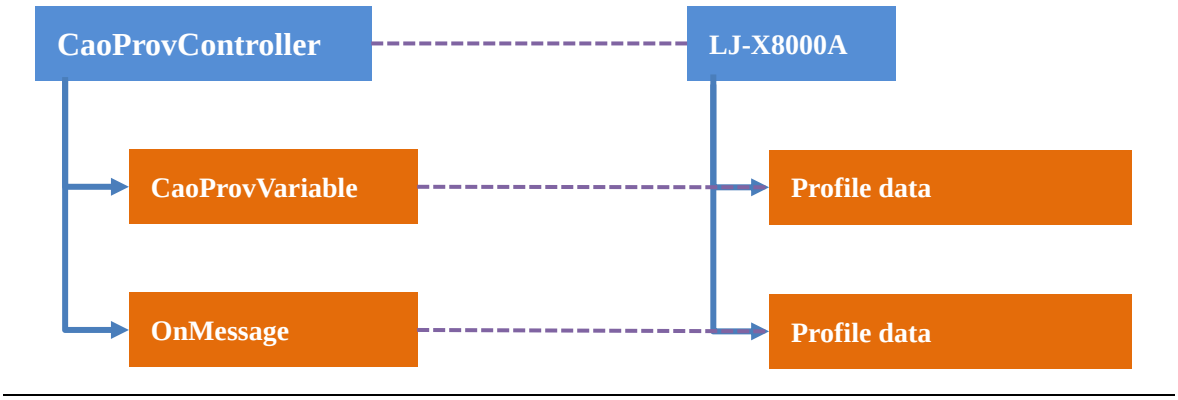


Fig. 1-2 Provider configuration and device information

2. Command Reference

2.1. Method/Property List

Table 2-1 List of methods and properties

Category	Methods/Properties[1]		Function	See Also
CaoWorkspace				
	AddController	M	Connected to controller	P.8
CaoController				
	VariableNames	P	Get a list of variable names that can be connected	P.10
	Variables	P	Retrieving Variable Collections Holded by the Controller	P.11
	AddVariable	M	Adding Variable Objects	P.12
	Execute	M	Execute Extended Commands	P.12
	OnMessage	E	Message reception event	P.27
CaoVariable				
	Value	P	Get/set value	P.27

2.2. Method properties

2.2.1. CaoWorkspace classes

2.2.1.1. AddController method

In CaoWorkspace, add a controller object. LJ-X8000A providers connect to the appropriate LJ-X8000A controllers by referring to the parameters passed when AddController method is executed. The following are the specifics of AddController method:

SYNOPSIS

AddController

```
(
        "<controller name>",                // Controller name (optional)
        "CaoProv.KEYENCE.LJ-X8000A",        // Provider name (fixed)
        "<machine name>",                   // Provider execution machine name (unused)
        "<Option>"                          // Option character string (optional)
)
```

[1] M: Indicates methods, P: properties, and E: events, respectively.

Option

The following is an optional specification for Option character string: Option character string is a comma (,) string consisting of the options listed below.

Option	Required	Description
Conn = <communication parameter>	✓	Specify the connection destination information.
DeviceID=<ID>	✓	Sets a unique ID for the controller to be connected. Set the ID within the range of 0 to 5. When connecting multiple IDs, make sure that there are no duplicates. e.g., "DeviceID=1"
HeadType = <head type>	--	Specify the type of head to be connected to the controller. X: LJ-X head V: LJ-V head
HSDComm = <high-speed data communication ON/OFF> [:<port number for high-speed data communication>: <number of profiles to send collectively at one time>]	--	Specifies whether to initiate high-speed data communication during AddController. When high-speed data communication is OFF (the first parameter is 0), the second and subsequent parameters can be omitted. (Default: 0) <High-speed data communication ON/OFF>: 0: OFF 1: ON For example: "HSDComm=0" "HSDComm=1:24691:10"
HSDCommEndMess=	--	Specifies whether to send a message at the end of high-speed data communication. 0: No message sent 1: Message sent

Usage example

```
// Engine
ORiN2.ManagedCAO.CCaoEngine engine = new ORiN2.ManagedCAO.CCaoEngine();
// Workspace
ORiN2.ManagedCAO.CCaoWorkspace workspace = engine.AddWorkspace("NewWrks", "");
// Controller
ORiN2.ManagedCAO.CCaoController controller= workspace.AddController("LJ-X8000A
                                "CaoProv.KEYENCE.LJ-X8000A",
                                "",
                                "Conn = ETH:192.168.10.10,DeviceId=1,HeadType=X,Timeout = 1000");
```

2.2.1.1.1. CONN Optional

The following is a Conn optional connection parameter string: Here, brackets ("[]") are optional, and the underlined part in the description of each parameter indicates the default value when no options are specified.

Connecting with a TCP/IP

"Conn = ETH: <destination IP>[:<destination port>[:<local IP>[:<local port>]]]]"

<Destination IP> : Specifies the destination IP address in the format ***.***.***.***.
Be sure to specify this item.

<Connection destination port> : Specify the port number to connect to. 24691

2.2.1.1.2. HSDComm option guidelines

Depending on the setting value, the number of profiles to be sent in one batch may run out of memory.

The operating environment verifies that 30000 profiles can be received at the same time, but the allowable range varies depending on the execution environment.

2.2.2. CaoController classes

2.2.2.1. VariableNames Properties

Gets a list of variable names that can be connected. The variable name obtained by this property can be used as the first argument of AddVariable method described later.

AddVariable method

Usage example

```
// Get variable name list
string[] variableNames = controller.GetVariableNames("");
```

2.2.2.2. Variables Properties

Gets a collection of variables that the controller holds.

Usage example

```
// Variable Collection Retrieval
ORiN2.ManagedCAO.CCaoVariables variables = controller.Variables;
// Variable acquisition
ORiN2.ManagedCAO.CCaoVariable variable = variables[0];
```

2.2.2.3. AddVariable method

Adds a variable object to CaoController. Only the variable names shown in 2.3.1 can be used.

AddVariable is specified as follows.

SYNOPSIS

AddVariable

```
(
        "<variable name>",          // Variable name
        "<Option>"                  // Option character string (optional)
)
```

2.2.2.4. Execute method

Execute CaoController extension. Execute is specifi ed as follows.

SYNOPSIS

Execute

```
(
        "<extension command name>",         // Extended command name
        "<Option string>"                   // Option character string (optional)
)
```

The following is a list of extended commands that can be specified in Execute. The usage examples are described in detail in the extended commands.

Command	Description	See Also
System control command		
GetError	Retrieves the specified number of error codes that have occurred in the controller.	P.13
ClearError	Clears the specified error code.	P.14
TrgErrorReset	Clears the ON state of TRG_ERROR.	P.14
GetTriggerAndPulseCount	Gets the trigger count and pulse count.	P.15
GetHeadTemperture	Gets the head temperature.	P.15
GetSerialNumber	Get the serial number.	P.16
GetAttentionStatus	Gets TRG_ERROR/MEM_FULL/TRG_PASS status	P.16

Command	Description	See Also
GetLedLightImage	Gets the LED illumination image.	P.16
Measurement Control Commands		
Trigger	Issue the trigger.	P.17
StartMesure	Starts the batch measurement.	P.17
StopMesure	Ends the batch measurement.	P.18
ClearMemory	Clears the data accumulated in the controller.	P.18
Setting Change/Read Related Commands		
GetActiveProgram	Gets the current active program number.	P.18
ChangeActiveProgram	Switches the active program number.	P.18
Measurement result acquisition command		
GetProfile	Get profile data	P.19
GetBatchProfile	Retrieves batch profile data.	P.20
GetbatchSimpleArray	Retrieves batch profile data (SimpleArray).	P.22
SetProfileCount	Set the number of profile data in one communication.	P.24
High-speed data communication related commands		
StartHighSpeedDataCommunication	Starts high-speed data communication.	P.26
StartHighSpeedDataCommunicationSimpleArray	Starts high-speed data communication (SimpleArray).	P.26
StopHighSpeedDataCommunication	Terminates high-speed data communication.	P.26

2.2.2.4.1. GetError Commands

Gets the number of errors and the specified number of error codes. By specifying 0 as the argument, only the number of errors that have occurred can be acquired. The following are the arguments and return values:

Item	Type Description		
Argument	VT_UI1		Specifies the number of errors to retrieve.
Return Value	VT_ARRAY | VT_VARIANT		
	0	VT_UI1	Gets the number of errors in the controller that occurred.
	1	VT_ARRAY | VT_UI2	

Item	Type Description			
		1. N	VT_UI2	Retrieves the error code number for the specified number of errors. If there are no specified number of errors in the controller, 0 for the specified number of errors is entered.

Usage example

```
// Get two error codes
object[] val = controller.Execute("GetError", 2) as object[];

int? errorNum = val[0] as int?; // Get error count
int?[] error = val[1] as int?[]; //Retrieved error code
```

2.2.2.4.2. ClearError Commands

Clears the system error of the specified error code. LJ-X8000A controllers start measuring when all errors have been cleared. The following are the arguments and return values:

Item	Type Description	
Argument	VT_UI2	Specify the error code to be cleared.[2]
Return Value	Nothing in particular	

Usage example

```
// Clear the error code 133.
controller.Execute("GetError", 133);
```

2.2.2.4.3. TrgErrorReset Commands

Clears the ON status of TRG_ERROR. TRG_ERROR is turned ON when parallel imaging is ON and the trigger interval is 10ms or longer. To check the TRG_ERROR status, check GetAttentionStatus. The following are the arguments and return values:

Item	Type Description
Argument	Nothing in particular.
Return Value	Nothing in particular.

Usage example

```
//Cancels TRG_ERRORON status.
controller.Execute("TrgErrorReset");
```

[2] Some error codes cannot be deleted.

2.2.2.4.4. GetTriggerAndPulseCount Commands

Gets the trigger count value and pulse count value. The following are the arguments and return values:

Item	Type Description		
Argument	Nothing in particular		
Return Value	VT_ARRAY | VT_VARIANT		
	0	VT_UI4	Gets the trigger count value. The count value range is 0 to 4,294,967,295 and returns to 0 when the count value exceeds the upper limit.
	1	VT_I4	Gets the pulse count value. The count value range is from-2,147,483,648 to 2,147,483,647, which is the maximum value of the negative upper limit when the positive upper limit is exceeded, and the maximum value of the positive upper limit when the negative upper limit is exceeded.

Usage example

```
// Get the trigger count pulse count
object[] val = controller.Execute("GetTriggerAndPulseCount", 2) as object[];

uint? triggerCount = val[0] as uint?; // Get trigger count
int? pulseCount = val[1] as int?;          // Acquire pulse count
```

2.2.2.4.5. GetHeadTemperature Commands

Reads the head temperature. The following are the arguments and return values:

Item	Type Description		
Argument	Nothing in particular		
Return Value	VT_ARRAY | VT_I2		
	0	VT_I2	Obtains the sensor (CMOS) temperature.
	1	VT_I2	Gets the processor temperature.
	2	VT_I2	Obtains the case temperature (enclosure). Since this field cannot be acquired by LJ-V head, the value of (0xFFFF) is acquired.

Usage example

```
// Get the head temperature
short?[] val = controller.Execute("GetHeadTempature") as short?[];
```

2.2.2.4.6. GetSerialNumber Commands

Gets the serial number of the controller and the head. The following are the arguments and return values:

Item	Type Description		
Argument	Nothing in particular		
Return Value	VT_ARRAY | VT_BSTR		
	0	VT_BSTR	Get the serial number of the controller.
	1	VT_BSTR	Gets the serial number of the head.

Usage example

```
// Obtain the serial number
object[] val = controller.Execute("GetSerialNumber") as object[];

string controllerSerial = val[0] as string; // Get controller serial number
string headSerial = val[1] as string;       // Get header serial number
```

2.2.2.4.7. GetAttentionStatus Commands

Gets the status of TRG_ERROR/MEM_FULL/TRG_PASS. The following are the arguments and return values:

Item	Type Description		
Argument	Nothing in particular		
Return Value	VT_ARRAY | VT_UI1		
	0	VT_UI1	TRG_ERROR state
	1	VT_UI1	MEM_FULL state
	2	VT_UI1	TRG_PASS state

Usage example

```
// Gets TRG_ERROR/MEM_FULL/TRG_PASS status.
object[] val = controller.Execute("GetAttentionStatus") as object[];

byte? trgErrror = val[0] as byte?; // Get TRG_ERROR status
byte? memFull = val[1] as byte?; // Get MEM_FULL status
byte? trgPass = val[2] as byte?; // Get TRG_PASS status
```

2.2.2.4.8. GetLedLightImage Commands

This function acquires an LED illumination image of 1-pixel 8-bit VGA (640 x 480) size with the specified LED and laser brightness. This command returns an error from the device when the LASER_ON pin is OFF. The following are the arguments and return values:

Item	Type Description

Item	Type Description		
Argument	VT_ARRAY | VT_UI1		
	0	VT_UI1	Specifies the brightness of the LEDs. Setting range: 1 to 100
	1	VT_UI1	Specifies the brightness of the laser. Setting range: 1 to 100
Return Value	VT_ARRAY | VT_UI1		
	N	VT_UI1	Gets the byte array of the LED illumination image.

Usage example

```
// Get LED illumination image
object[] val = controller.Execute("GetAttentionStatus") as object[];

byte?[] ledLightImage = val[0] as byte?[]; // Get bytes of LEDs
```

2.2.2.4.9. Trigger Commands

Issue the trigger. In order for the controller to execute a trigger command from the provider, it must be set to external trigger input in the controller's configuration. The following are the arguments and return values:

Item	Type Description
Argument	Nothing in particular.
Return Value	Nothing in particular.

Usage example

```
//Issue a trigger
controller.Execute("Trigger");
```

2.2.2.4.10. StartMeasure Commands

Starts the batch measurement. If batch measurement is OFF or the multiple controller synchronization function is used, but the specified device is not set to "Synchronization master", an error is returned from the device when the LASER_ON pin is OFF. The following are the arguments and return values:

Item	Type Description
Argument	Nothing in particular.
Return Value	Nothing in particular.

Usage example

```
// Start batch measurements
controller.Execute("StartMeasure");
```

2.2.2.4.11. StopMeasure Commands

Ends the batch measurement. If batch measurement is OFF or the multiple controller synchronization function is used, but the specified device is not set to "Synchronization master", an error is returned from the device when the LASER_ON pin is OFF. The following are the arguments and return values:

Item	Type Description
Argument	Nothing in particular.
Return Value	Nothing in particular.

Usage example

```
//end batch measurement
controller.Execute("StopMeasure");
```

2.2.2.4.12. ClearMemory Commands

Clears the profile data stored in the controller. The following are the arguments and return values:

Item	Type Description
Argument	Nothing in particular.
Return Value	Nothing in particular.

Usage example

```
//Clears the data stored in the controller.
controller.Execute("ClearMemory");
```

2.2.2.4.13. GetActiveProgram Commands

Gets the current active program number. The following are the arguments and return values:

Item	Type Description	
Argument	Nothing in particular.	
Return Value	VT_UI1	Acquires the active program number.

Usage example

```
//Gets the current active program number.
byte? activeProgram = controller.Execute("GetActiveProgram") as byte?;
```

2.2.2.4.14. ChangeActiveProgram Commands

Switches the active program number. The following are the arguments and return values:

Item	Type Description	
Argument	VT_UI1	Specify the active program number after switching.
Return Value	Nothing in particular.	

Usage example

```
//toggles the active program number to 5.
byte activeProgramNo = 5;
controller.Execute("ChangeActiveProgram", activeProgramNo);
```

2.2.2.4.15. GetProfile Commands

Get profile data. The following are the arguments and return values: For details about the various items of argument specification values, see "LJ-X8000A Communication Library Reference Manual".

<table>
<tr><th>Item</th><th colspan="4">Type Description</th></tr>
<tr><td rowspan="6">Argument</td><td colspan="4">VT_ARRAY | VT_VARIANT</td></tr>
<tr><td>0</td><td>VT_UI1</td><td colspan="2">Specifies whether to acquire from an active or inactive plane.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>Active plane</td></tr><tr><td>1</td><td>Inactive plane</td></tr></table></td></tr>
<tr><td>1</td><td>VT_UI1</td><td colspan="2">Specifies how to specify the profile acquisition location.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>From the latest</td></tr><tr><td>1</td><td>From the oldest</td></tr><tr><td>2</td><td>Arbitrary position specification</td></tr></table></td></tr>
<tr><td>2</td><td>VT_UI4</td><td colspan="2">When 2 is specified in the profile position specification method, specify the profile number to be acquired.
Value range: 0 to 4294967295
※If the specified profile number does not exist, an error is returned.</td></tr>
<tr><td>3</td><td>VT_UI1</td><td colspan="2">Specify the number of profiles to read.
Value range: 1 to 255</td></tr>
<tr><td>4</td><td>VT_UI1</td><td colspan="2">Specify whether to erase the read profile and the previous profile.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>Without deleting</td></tr><tr><td>1</td><td>To delete</td></tr></table></td></tr>
<tr><td rowspan="7">Return Value</td><td colspan="4">VT_ARRAY | VT_VARIANT</td></tr>
<tr><td rowspan="6">0</td><td colspan="3">VT_ARRAY | VT_VARIANT</td></tr>
<tr><td>0.0</td><td>VT_UI1</td><td>Gets the number of profiles stored in one unit of data.</td></tr>
<tr><td>0.1</td><td>VT_UI1</td><td>Gets ON/OFF of brightness.</td></tr>
<tr><td>0.2</td><td>VT_UI2</td><td>Gets the number of data points in one profile.</td></tr>
<tr><td>0.3</td><td>VT_I4</td><td>Gets the X coordinate of the first point.</td></tr>
<tr><td>0.4</td><td>VT_I4</td><td>X spacing of data points</td></tr>
</table>

Item	Type Description			
		0.5	VT_I4	Gets the latest profile number at the time of acquisition.
		0.6	VT_UI4	Acquires the oldest profile number held by the controller.
		0.7	VT_UI4	Acquires the oldest profile number that has been read.
		0.8	VT_UI1	Gets the number of profiles read this time.
	1. N	VT_ARRAY|VT_I4		Acquires the read profile data. The acquired profile data is stored in order.

Usage example

```
// Gets one profile data from the 10th profile data
object[] prm = new object[]{0,2,10,1,0};
object[] retValue = controller.Execute("GetProfile",prm) as object[];

object[] profileInfo = retValue[0] as object[]; // Profile info

byte? ProfileNum = profileInfo [0] as byte?; // Number of profiles
byte? Luminication = profileInfo [1] as byte?; // brightness information
short? ProfileDataCount profileInfo [2] as short?; // Number of profile data points
int? XPos = profileInfo [3] as int?; // X-coordinate
int? XInterval = profileInfo [4] as int?; // X spacing
uint? CurrentProfileNo = profileInfo [5] as uint?; // Latest profile number
uint? OldestProfileNo = profileInfo [6] as uint?; // Oldest profile number
byte? GetProfileCount = profileInfo [7] as byte?; // Number of profiles acquired this time

int?[] profileData = retValue[1] as int[]?; // Profile data
```

2.2.2.4.16. GetBatchProfile Commands

Retrieves batch profile data. The following are the arguments and return values:

Item	Type Description		
Argument	VT_ARRAY | VT_VARIANT		
	0	VT_UI1	Specifies whether to acquire from an active or inactive plane.

Set value	Introduction
0	Active plane
1	Inactive plane

1	VT_UI1	Specifies how to specify the batch acquisition position.

Set value	Introduction
0	From the latest
2	Arbitrary position specification
3	From the most recent time after the batch is confirmed
4	Latest only

<table>
<tr><th>Item</th><th colspan="4">Type Description</th></tr>
<tr><td rowspan="5"></td><td>2</td><td>VT_UI4</td><td colspan="2">If 2 is specified in the batch acquisition position specification method, specify the batch number to be acquired.
Value range: 0 to 4294967295
※If the specified batch number does not exist, an error is returned.</td></tr>
<tr><td>3</td><td>VT_UI4</td><td colspan="2">Specifies the acquisition start profile number in the batch.
Value range: 0 to 4294967295
※If the specified profile number does not exist, an error is returned.</td></tr>
<tr><td>4</td><td>VT_UI1</td><td colspan="2">Specify the number of profiles to read.
Value range: 1 to 255</td></tr>
<tr><td>5</td><td>VT_UI1</td><td colspan="2">Specify whether to erase the read profile and the previous profile.
<table>
<tr><th>Set value</th><th>Introduction</th></tr>
<tr><td>0</td><td>Without deleting</td></tr>
<tr><td>1</td><td>To delete</td></tr>
</table></td></tr>
<tr><td></td><td colspan="4"></td></tr>
<tr><td rowspan="16">Return Value</td><td colspan="4">VT_ARRAY | VT_VARIANT</td></tr>
<tr><td rowspan="15">0</td><td colspan="4">VT_ARRAY | VT_VARIANT</td></tr>
<tr><td>0.0</td><td>VT_UI1</td><td colspan="2">Gets the number of profiles stored in one unit of data.</td></tr>
<tr><td>0.1</td><td>VT_UI1</td><td colspan="2">Gets ON/OFF of brightness.</td></tr>
<tr><td>0.2</td><td>VT_UI2</td><td colspan="2">Gets the number of data points in one profile.</td></tr>
<tr><td>0.3</td><td>VT_I4</td><td colspan="2">Gets the X coordinate of the first point.</td></tr>
<tr><td>0.4</td><td>VT_I4</td><td colspan="2">X spacing of data points</td></tr>
<tr><td>0.5</td><td>VT_I4</td><td colspan="2">Gets the latest profile number at the time of acquisition.</td></tr>
<tr><td>0.6</td><td>VT_UI4</td><td colspan="2">Acquires the oldest profile number held by the controller.</td></tr>
<tr><td>0.7</td><td>VT_UI4</td><td colspan="2">Retrieves the number of profiles in the oldest batch held by the controller.</td></tr>
<tr><td>0.8</td><td>VT_UI4</td><td colspan="2">Acquires the oldest profile number that has been read.</td></tr>
<tr><td>0.9</td><td>VT_UI4</td><td colspan="2">Acquires the batch number read this time.</td></tr>
<tr><td>0.10</td><td>VT_UI4</td><td colspan="2">Gets the number of profiles in the batch read this time.</td></tr>
<tr><td>0.11</td><td>VT_UI4</td><td colspan="2">Gets the number of profiles in the read batch.</td></tr>
<tr><td>0.12</td><td>VT_UI1</td><td colspan="2">Gets the number of profiles read this time.</td></tr>
<tr><td>0.13</td><td>VT_UI1</td><td colspan="2">Acquires whether or not the latest batch measurement has been completed.</td></tr>
<tr><td>1. N</td><td colspan="2">VT_ARRAY|VT_I4</td><td colspan="2">Acquires the read profile data.
The acquired profile data is stored in order.</td></tr>
</table>

Usage example

```
// Get one profile data from 1st to 10th profile data in batch
object[] prm = new object[]{0,2,1,10,1,0};
object[] retValue = controller.Execute("GetBatchProfile",prm) as object[];

object[] profileInfo = retValue[0] as object[]; // Profile info

byte? ProfileNum = profileInfo [0] as byte?; // Number of profiles
byte? Luminication = profileInfo [1] as byte?; // brightness information
short? ProfileDataCount profileInfo [2] as short?; // Number of profile data points
int? XPos = profileInfo [3] as int?; // X-coordinate
int? XInterval = profileInfo [4] as int?; // X spacing
uint? CurrentProfileNo = profileInfo [5] as uint?; // Latest profile number
uint? OldestProfileNo = profileInfo [6] as uint?; // Oldest profile number
uint? GetProfileCount = profileInfo [7] as uint?; // Number of profiles in the oldest batch
uint? GetOldestProfNo = profileInfo [8] as uint?; //oldest profile number read
uint? GetBatchNo = profileInfo [9] as uint?; // Batch number read
uint? GetProfileNo = profileInfo [10] as uint?; // Number of profiles in batches read
uint? GetBatchProfileNo = profileInfo [11] as uint?; // Which profile is
byte? GetProfileCount = profileInfo [12] as uint?; //oldest profile number read
byte? isBatchMesure = profileInfo [13] as byte?; //Whether the batch test has been completed

int?[] profileData = retValue[1] as int[]?; // Profile data
```

2.2.2.4.17. GetBatchSimpleArray Commands

Retrieves batch profile data in SimpleArray. The following are the arguments and return values:

<table>
<tr><th>Item</th><th colspan="3">Type Description</th></tr>
<tr><td rowspan="5">Argument</td><td colspan="3">VT_ARRAY | VT_VARIANT</td></tr>
<tr><td>0</td><td>VT_UI1</td><td>Specifies whether to acquire from an active or inactive plane.
<table>
<tr><th>Set value</th><th>Introduction</th></tr>
<tr><td>0</td><td>Active plane</td></tr>
<tr><td>1</td><td>Inactive plane</td></tr>
</table></td></tr>
<tr><td>1</td><td>VT_UI1</td><td>Specifies how to specify the batch acquisition position.
<table>
<tr><th>Set value</th><th>Introduction</th></tr>
<tr><td>0</td><td>From the latest</td></tr>
<tr><td>2</td><td>Arbitrary position specification</td></tr>
<tr><td>3</td><td>From the most recent time after the batch is confirmed</td></tr>
<tr><td>4</td><td>Latest only</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Item</th><th colspan="5">Type Description</th></tr>
<tr><td rowspan="5"></td><td>2</td><td>VT_UI4</td><td colspan="3">If 2 is specified in the batch acquisition position specification method, specify the batch number to be acquired.
Value range: 0 to 4294967295
※If the specified batch number does not exist, an error is returned.</td></tr>
<tr><td>3</td><td>VT_UI4</td><td colspan="3">Specifies the acquisition start profile number in the batch.
Value range: 0 to 4294967295
※If the specified profile number does not exist, an error is returned.</td></tr>
<tr><td>4</td><td>VT_UI1</td><td colspan="3">Specify the number of profiles to read.
Value range: 1 to 255</td></tr>
<tr><td>5</td><td>VT_UI1</td><td colspan="3">Specify whether to erase the read profile and the previous profile.
<table>
<tr><th>Set value</th><th>Introduction</th></tr>
<tr><td>0</td><td>Without deleting</td></tr>
<tr><td>1</td><td>To delete</td></tr>
</table></td></tr>
<tr><td colspan="5"></td></tr>
<tr><td rowspan="18">Return Value</td><td colspan="5">VT_ARRAY | VT_VARIANT</td></tr>
<tr><td>0</td><td colspan="4">VT_ARRAY | VT_VARIANT</td></tr>
<tr><td></td><td>0.0</td><td>VT_UI1</td><td colspan="2">Gets the number of profiles stored in one unit of data.</td></tr>
<tr><td></td><td>0.1</td><td>VT_UI1</td><td colspan="2">Gets ON/OFF of brightness.</td></tr>
<tr><td></td><td>0.2</td><td>VT_UI2</td><td colspan="2">Gets the number of data points in one profile.</td></tr>
<tr><td></td><td>0.3</td><td>VT_I4</td><td colspan="2">Gets the X coordinate of the first point.</td></tr>
<tr><td></td><td>0.4</td><td>VT_I4</td><td colspan="2">X spacing of data points</td></tr>
<tr><td></td><td>0.5</td><td>VT_I4</td><td colspan="2">Gets the latest profile number at the time of acquisition.</td></tr>
<tr><td></td><td>0.6</td><td>VT_UI4</td><td colspan="2">Acquires the oldest profile number held by the controller.</td></tr>
<tr><td></td><td>0.7</td><td>VT_UI4</td><td colspan="2">Retrieves the number of profiles in the oldest batch held by the controller.</td></tr>
<tr><td></td><td>0.8</td><td>VT_UI4</td><td colspan="2">Acquires the oldest profile number that has been read.</td></tr>
<tr><td></td><td>0.9</td><td>VT_UI1</td><td colspan="2">Acquires the batch number read this time.</td></tr>
<tr><td></td><td>0.10</td><td>VT_UI4</td><td colspan="2">Gets the number of profiles in the batch read this time.</td></tr>
<tr><td></td><td>0.11</td><td>VT_UI4</td><td colspan="2">Gets the oldest profile read out and the number of profiles in the batch.</td></tr>
<tr><td></td><td>0.12</td><td>VT_UI1</td><td colspan="2">Gets the number of profiles read this time.</td></tr>
<tr><td></td><td>0.13</td><td>VT_UI1</td><td colspan="2">Acquires whether or not the latest batch measurement has been completed.</td></tr>
<tr><td>1. N</td><td colspan="2">VT_ARRAY|VT_I4</td><td colspan="2">The header information of the read profile data is acquired.</td></tr>
<tr><td>2. N</td><td colspan="2">VT_ARRAY|VT_UI2</td><td colspan="2">The height information of the read profile data is acquired.</td></tr>
</table>

Item	Type Description		
	3. N	VT_ARRAY|VT_UI2	Brightness information of the read profile data is acquired.

Usage example

```
// Gets one profile data from the 10th profile data
object[] prm = new object[]{0,2,10,1,0};
object[] retValue = controller.Execute("GetBatchProfileSimpleArray",prm) as object[];

object[] profileInfo = retValue[0] as object[]; // Profile info

byte? ProfileNum = profileInfo [0] as byte?; // Number of profiles
byte? Luminication = profileInfo [1] as byte?; // brightness information
ushort? ProfileDataCount profileInfo [2] as ushort?; // Number of profile data points
int? XPos = profileInfo [3] as int?; // X-coordinate
int? XInterval = profileInfo [4] as int?; // X spacing
uint? CurrentProfileNo = profileInfo [5] as uint?; // Latest profile number
uint? OldestProfileNo = profileInfo [6] as uint?; // Oldest profile number
uint? GetProfileCount = profileInfo [7] as uint?; // Number of profiles in the oldest batch
uint? GetOldestProfNo = profileInfo [8] as uint?; //oldest profile number read
uint? GetBatchNo = profileInfo [9] as uint?; // Batch number read
uint? GetProfileNo = profileInfo [10] as uint?; // Number of profiles in batches read
uint? GetBatchProfileNo = profileInfo [11] as uint?; // Which profile is
byte? GetProfileCount = profileInfo [12] as uint?; //oldest profile number read
byte? isBatchMesure = profileInfo [13] as byte?; //Whether the batch test has been completed

int?[] profileData = retValue[1] as int[]?; // profile header data
ushort?[] profileData = retValue[2] as ushort[]?; // Profile height data
ushort?[] profileData = retValue[3] as ushort[]?; // Profile brightness data
```

2.2.2.4.18. SetProfileCount Commands

Specifies the buffer size to pass to the API when retrieving profile data.

This provider uses the API of the library to acquire profile data. When executing the API, it is necessary to allocate memory for the number of profile data to be acquired beforehand, and the provider has allocated a maximum memory buffer (25628 bytes per profile) in the initial state. It consists of height data (3200 points × 4 bytes), brightness data (3200 × 4 bytes), profile data header (24 bytes), and profile data footer (4 bytes). If the data cannot be retrieved due to memory shortage, use this command to reduce the buffer size. If the buffer size is reduced, it must be set so that it is the same size on the controller side as well.

※When the head used in the setting on the controller is LJX series measuring range = 1/2, thinned-out X axis = FULL, and sampling period =16kHz, 3200 (number of LJX head reference profile points) x 0.5 x 1.0 x 0.5 = 800.

Setting (2,0,1) as the argument of the command results in the same value.

Item	Type Description
Argument	VT_ARRAY | VT_UI1

<table>
<tr><td>Item</td><td colspan="3">Type Description</td></tr>
<tr><td></td><td>0</td><td>VT_UI1</td><td>Specifies the measurement range X direction.

<table>
<tr><th>Specified value</th><th>Controller side set value</th><th>Coefficient</th></tr>
<tr><td>0</td><td>OFF</td><td>1.0</td></tr>
<tr><td>1</td><td>3/4</td><td>0.75</td></tr>
<tr><td>2</td><td>1/2</td><td>0.5</td></tr>
<tr><td>3</td><td>1/4</td><td>0.25</td></tr>
</table></td></tr>
<tr><td></td><td>1</td><td>VT_UI1</td><td>Specify the skipping X axis.

<table>
<tr><th>Specified value</th><th>Controller side set value</th><th>Coefficient</th></tr>
<tr><td>0</td><td>OFF</td><td>1.0</td></tr>
<tr><td>1</td><td>1/2</td><td>0.5</td></tr>
<tr><td>2</td><td>1/4</td><td>0.25</td></tr>
</table></td></tr>
<tr><td></td><td>2</td><td>VT_UI1</td><td>The items specified for the heads of LJ-X series differ from those of LJ-V series.

LJ-X head: Sampling period
<table>
<tr><th>Specified value</th><th>Controller side set value</th><th>Coefficient</th></tr>
<tr><td>0</td><td>Settings other than 8 kHz or 16kHz</td><td>1.0</td></tr>
<tr><td>1</td><td>8kHz,16kHz</td><td>0.5</td></tr>
</table>
LJ-V head: binning
<table>
<tr><th>Specified value</th><th>Controller side set value</th><th>Coefficient</th></tr>
<tr><td>0</td><td>OFF</td><td>1.0</td></tr>
<tr><td>1</td><td>ON</td><td>0.5</td></tr>
</table></td></tr>
<tr><td>Return Value</td><td colspan="3">Nothing in particular.</td></tr>
</table>

Usage example

```
// 800 data points at the LJX head
// (height (800 × 4 bytes) + brightness (800 × 4 bytes) + header (24) + footer (4) = 6428 bytes
object[] prm = new object[]{2,1,0};
controller.Execute("SetProfileCount ", prm);
```

2.2.2.4.19. StartHighSpeedDataCommunication Commands

Starts high-speed data communication. The following are the arguments and return values:

Item	Type Description		
Argument	VT_ARRAY | VT_VARIANT		
	0	VT_UI2	Port number used for high-speed data
	1	VT_UI4	Number of profiles to send collectively An OnMessage occurs when a specified number of profiles are received.
Return Value	Nothing in particular.		

Usage example

```
// Start high-speed data communication.
object[] prm = new object[]{28461, 10};
controller.Execute("StartHighSpeedDataCommunication", prm);
```

2.2.2.4.20. StartHighSpeedDataCommunicationSimpleArray Commands

Starts high-speed data communication in SimpleArray. The following are the arguments and return values:

Item	Type Description		
Argument	VT_ARRAY | VT_VARIANT		
	0	VT_UI2	Port number used for high-speed data
	1	VT_UI4	Number of profiles to send collectively An OnMessage occurs when a specified number of profiles are received.
Return Value	Nothing in particular.		

Usage example

```
// Starts high-speed data communication in SimpleArray.
object[] prm = new object[]{28461, 10};
controller.Execute("StartHighSpeedDataCommunicationSimpleArray", prm);
```

2.2.2.4.21. StopHighSpeedDataCommunication Commands

Terminates high-speed data communication. The following are the arguments and return values:

Item	Type Description
Argument	Nothing in particular.
Return Value	Nothing in particular.

Usage example

```
// Terminates high-speed data communication.
controller.Execute("StopHighSpeedDataCommunication ");
```

2.2.2.5. OnMessage event

You can receive controller error notifications and status changes as OnMessage events. See 2.4 for the events that can be received.

2.2.3. CaoVariable classes

2.2.3.1. Value Properties

The behavior depends on the variable name. For details, refer to section 2.3, Variable List.

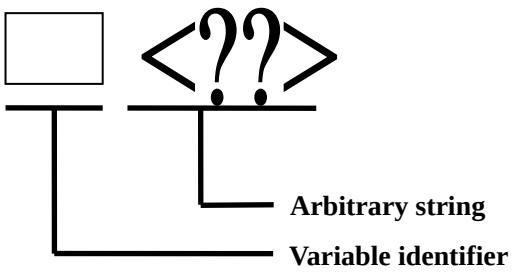
2.3. Variable list

Defines a list of variables that can be used in each class. Variables refer to objects of CaoVariable classes.

Variables refer to objects of CaoVariable classes. An arbitrary string can be added to register multiple variables (useful when changing only options, etc.).

The following format gives an arbitrary string to a variable name:

Multi-variable common specification format



2.3.1. CaoController class-variable

Variable name	Description	Value		See Also
		Get	Put	
@MAKER_NAME	Obtain the manufacturer's name.	✓	-	P.28
@VERSION	Get the DLL version.	✓	-	P.28
CURRENT_PROFILE<??>	Retrieves profile data from the latest.	✓	-	P.29
OLDEST_PROFILE<??>	Retrieves profile data from the oldest.	✓	-	P.30
SPEC_PROFILE<??>	Retrieves profile data by specifying the position.	✓	-	P.31
CURRENT_BATCHPROFILE<??>	Retrieve profile data from the latest batch.	✓	-	P.32
SPEC_BATCHPROFILE<??>	Retrieves profile data from a specific batch by locating it.	✓	-	P.34

COMMITED_BATCHPROFILE<??>	Retrieves profile data from the latest batch after the batch is confirmed.	✓	-	P.35
CURRENTONLY_BATCHPROFILE<??>	Retrieves only the latest profile data from the latest batch.	✓	-	P.36
CURRENT_BATCHPROFILE_SIMPLE<??>	Retrieve profile data from the latest batch.	✓	-	P.37
SPEC_BATCHPROFILE_SIMPLE<??>	Retrieves profile data from a specific batch by locating it.	✓	-	P.39
COMMITED_BATCHPROFILE_SIMPLE<??>	Retrieves profile data from the latest batch after the batch is confirmed.	✓	-	P.40
CURRENTONLY_BATCHPROFILE_SIMPLE<??>	Retrieves only the latest profile data from the latest batch.	✓	-	P.41

2.3.1.1. @MAKER_NAME

Obtain the manufacturer's name.

Data Type

Type Description	
VT_BSTR	Obtain the manufacturer's name.

Usage example

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@MAKER_NAME","");
// Acquisition of Values
string value = var.Value as string;
```

2.3.1.2. @VERSION

Gets the DLL version.

Data Type

Type Description	
VT_BSTR	Get the DLL version. *.*.*

Usage example

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@VERSION","");
// Acquisition of Values
string value = var.Value as string;
```

2.3.1.3. CURRENT_PROFILE<??>

Retrieves the specified number of profile data from the latest. Retrieves the specified number of profile data by specifying the number of profile data to retrieve with the option.

The data to be retrieved is shown below.

Type Description			
VT_ARRAY | VT_VARIANT			
0	VT_ARRAY | VT_VARIANT		
	0.0	VT_UI1	Gets the number of profiles stored in one unit of data.
	0.1	VT_UI1	Gets ON/OFF of brightness.
	0.2	VT_UI2	Gets the number of data points in one profile.
	0.3	VT_I4	Gets the X coordinate of the first point.
	0.4	VT_I4	X spacing of data points
	0.5	VT_I4	Gets the latest profile number at the time of acquisition.
	0.6	VT_UI4	Acquires the oldest profile number held by the controller.
	0.7	VT_UI4	Acquires the oldest profile number that has been read.
	0.8	VT_UI1	Gets the number of profiles read this time.
1. N	VT_ARRAY|VT_I4		Acquires the read profile data. The acquired profile data is stored in order.

Option	Required	Description	Value Range
TargetBank = [=<acquisition plane>]	--	Specifies whether to acquire from an active or inactive plane. Set value: Introduction 0: Active plane 1: Inactive plane Default: 0	0-1
ProfileCount [=<number of profiles to retrieve>]	✓	Specify the number of profiles to read.	1-255
Erase = [<whether to delete profile data>]	--	Specify whether to erase the read profile and the previous profile. Set value: Introduction 0: Without deleting 1: To delete	0-1

Usage example

```
// The number of profiles to retrieve is one and variables are added.
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("CURRENT_PROFILE01","ProfileCount=1");

/Get profile data
object[] retValue = var.Value as object[];

object[] profileInfo = retValue[0] as object[]; // Profile info

byte? ProfileNum = profileInfo [0] as byte?; // Number of profiles
byte? Luminication = profileInfo [1] as byte?; // brightness information
short? ProfileDataCount profileInfo [2] as short?; // Number of profile data points
int? XPos = profileInfo [3] as int?; // X-coordinate
int? XInterval = profileInfo [4] as int?; // X spacing
uint? CurrentProfileNo = profileInfo [5] as uint?; // Latest profile number
uint? OldestProfileNo = profileInfo [6] as uint?; // Oldest profile number
byte? GetProfileCount = profileInfo [7] as byte?; // Number of profiles acquired this time

int?[] profileData = retValue[1] as int[]?; // Profile data
```

2.3.1.4. OLDEST_PROFILE<??>

Retrieves profile data from the oldest. Retrieves the specified number of profile data by specifying the number of profile data to retrieve with the option.

The data to be acquired is the same as 2.3.1.3, so please refer to it.

<table>
<tr><th>Option</th><th>Required</th><th>Description</th><th>Value Range</th></tr>
<tr><td>TargetBank = [=
<acquisition plane>]</td><td>--</td><td>Specifies whether to acquire from an active or inactive plane.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>Active plane</td></tr><tr><td>1</td><td>Inactive plane</td></tr></table>Default: 0</td><td>0-1</td></tr>
<tr><td>ProfileCount
[=<number of profiles to retrieve>]</td><td>✓</td><td>Specify the number of profiles to read.</td><td>1-255</td></tr>
<tr><td>Erase = [<whether to delete profile data>]</td><td>--</td><td>Specify whether to erase the read profile and the previous profile.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>Without deleting</td></tr><tr><td>1</td><td>To delete</td></tr></table></td><td>0-1</td></tr>
</table>

Usage example

```
// The number of profiles to retrieve is one and variables are added.
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("OLDEST_PROFILE01","ProfileCount=1
");

// Retrieve profile data
object[] retValue = var.Value as object[];

object[] profileInfo = retValue[0] as object[]; // Profile info

byte? ProfileNum = profileInfo [0] as byte?; // Number of profiles
byte? Luminication = profileInfo [1] as byte?; // brightness information
short? ProfileDataCount profileInfo [2] as short?; // Number of profile data points
int? XPos = profileInfo [3] as int?; // X-coordinate
int? XInterval = profileInfo [4] as int?; // X spacing
uint? CurrentProfileNo = profileInfo [5] as uint?; // Latest profile number
uint? OldestProfileNo = profileInfo [6] as uint?; // Oldest profile number
byte? GetProfileCount = profileInfo [7] as byte?; // Number of profiles acquired this time

int?[] profileData = retValue[1] as int[]?; // Profile data
```

2.3.1.5. SPEC_PROFILE<??>

Retrieves profile data from a specified location. Specifies the profile number to be acquired with the option, and acquires the specified number of profile data from that location.

The data to be acquired is the same as 2.3.1.3, so please refer to it.

<table>
<tr><th>Option</th><th>Required</th><th>Description</th><th>Value Range</th></tr>
<tr><td>TargetBank = [= <acquisition plane>]</td><td>--</td><td>Specifies whether to acquire from an active or inactive plane.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>Active plane</td></tr><tr><td>1</td><td>Inactive plane</td></tr></table>Default: 0</td><td>0-1</td></tr>
<tr><td>ProfileNo = [= <profile number to be acquired>]</td><td>✓</td><td>When 2 is specified in the profile position specification method, specify the profile number to be acquired.</td><td>0~4294967295</td></tr>
<tr><td>ProfileCount [=<number of profiles to retrieve>]</td><td>✓</td><td>Specify the number of profiles to read.</td><td>1-255</td></tr>
<tr><td>Erase = [<whether to delete profile data>]</td><td>--</td><td>Specify whether to erase the read profile and the previous profile.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>Without deleting</td></tr><tr><td>1</td><td>To delete</td></tr></table></td><td>0-1</td></tr>
</table>

Usage example

```
// Added a variable that gets one profile count from the profile No10
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("SPEC_PROFILE01","ProfileCount=1,Pro
fileNo=10");

// Retrieve profile data
object[] retValue = var.Value as object[];

object[] profileInfo = retValue[0] as object[]; // Profile info

byte? ProfileNum = profileInfo [0] as byte?; // Number of profiles
byte? Luminication = profileInfo [1] as byte?; // brightness information
short? ProfileDataCount profileInfo [2] as short?; // Number of profile data points
int? XPos = profileInfo [3] as int?; // X-coordinate
int? XInterval = profileInfo [4] as int?; // X spacing
uint? CurrentProfileNo = profileInfo [5] as uint?; // Latest profile number
uint? OldestProfileNo = profileInfo [6] as uint?; // Oldest profile number
byte? GetProfileCount = profileInfo [7] as byte?; // Number of profiles acquired this time

int?[] profileData = retValue[1] as int[]?; // Profile data
```

2.3.1.6. CURRENT_BATCHPROFILE<??>

Retrieves profile data from the latest. Retrieves the specified number of profile data by specifying the number of profile data to retrieve with the option.

The data to be retrieved is shown below.

Type Description			
VT_ARRAY | VT_VARIANT			
0	VT_ARRAY | VT_VARIANT		
	0.0	VT_UI1	Gets the number of profiles stored in one unit of data.
	0.1	VT_UI1	Gets ON/OFF of brightness.
	0.2	VT_UI2	Gets the number of data points in one profile.
	0.3	VT_I4	Gets the X coordinate of the first point.
	0.4	VT_I4	X spacing of data points
	0.5	VT_I4	Gets the latest profile number at the time of acquisition.
	0.6	VT_UI4	Acquires the oldest profile number held by the controller.
	0.7	VT_UI4	Retrieves the number of profiles in the oldest batch held by the controller.
	0.8	VT_UI4	Acquires the oldest profile number that has been read.
	0.9	VT_UI4	Acquires the batch number read this time.
	0.10	VT_UI4	Gets the number of profiles in the batch read this time.
	0.11	VT_UI4	Gets the number of profiles in the read batch.

	0.12	VT_UI1	Gets the number of profiles read this time.
	0.13	VT_UI1	Acquires whether or not the latest batch measurement has been completed.
1. N	VT_ARRAY|VT_I4		Acquires the read profile data. The acquired profile data is stored in order.

Option	Required	Description	Value Range
TargetBank = [=<acquisition plane>]	--	Specifies whether to acquire from an active or inactive plane. Set value: 0 = Active plane; 1 = Inactive plane Default: 0	0-1
ProfileCount [=<number of profiles to retrieve>]	✓	Specify the number of profiles to read.	1-255
Erase = [<whether to delete profile data>]	--	Specify whether to erase the read profile and the previous profile. Set value: 0 = Without deleting; 1 = To delete	0-1

Usage example

```
//add a variable to get one profile data
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("CURRENT_BATCHPROFILE01","ProfileCount=1");

// Retrieve profile data
object[] retValue = var.Value as object[];

object[] profileInfo = retValue[0] as object[]; // Profile info

byte? ProfileNum = profileInfo [0] as byte?; // Number of profiles
byte? Luminication = profileInfo [1] as byte?; // brightness information
short? ProfileDataCount profileInfo [2] as short?; // Number of profile data points
int? XPos = profileInfo [3] as int?; // X-coordinate
int? XInterval = profileInfo [4] as int?; // X spacing
uint? CurrentProfileNo = profileInfo [5] as uint?; // Latest profile number
uint? OldestProfileNo = profileInfo [6] as uint?; // Oldest profile number
uint? GetProfileCount = profileInfo [7] as uint?; // Number of profiles in the oldest batch
uint? GetOldestProfNo = profileInfo [8] as uint?; //oldest profile number read
uint? GetBatchNo = profileInfo [9] as uint?; // Batch number read
uint? GetProfileNo = profileInfo [10] as uint?; // Number of profiles in batches read
uint? GetBatchProfileNo = profileInfo [11] as uint?; // Which profile is
```

```
byte? GetProfileCount = profileInfo [12] as uint?; //oldest profile number read
byte? isBatchMesure = profileInfo [13] as byte?; //Whether the batch test has been completed

int?[] profileData = retValue[1] as int[]?; // Profile data
```

2.3.1.7. SPEC_BATCHPROFILE<??>

Retrieves profile data from a location that you specify. Specify the batch No. and profile No. to be acquired optionally, and obtain the specified number of profile data from the specified location.

The data to be acquired is the same as 2.3.1.6, so please refer to it.

<table>
<tr><th>Option</th><th>Required</th><th>Description</th><th>Value Range</th></tr>
<tr><td>TargetBank = [= <acquisition plane>]</td><td>--</td><td>Specifies whether to acquire from an active or inactive plane.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>Active plane</td></tr><tr><td>1</td><td>Inactive plane</td></tr></table>Default: 0</td><td>0-1</td></tr>
<tr><td>BatchNo = [= <Batch No>]</td><td>✓</td><td>Specifies the batch number for which profile data is to be retrieved.</td><td>0~4294967295</td></tr>
<tr><td>ProfileNo = [= <profile number to be acquired>]</td><td>✓</td><td>When 2 is specified in the profile position specification method, specify the profile number to be acquired.</td><td>0~4294967295</td></tr>
<tr><td>ProfileCount [=<number of profiles to retrieve>]</td><td>✓</td><td>Specify the number of profiles to read.</td><td>1-255</td></tr>
<tr><td>Erase = [<whether to delete profile data>]</td><td>--</td><td>Specify whether to erase the read profile and the previous profile.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>Without deleting</td></tr><tr><td>1</td><td>To delete</td></tr></table></td><td>0-1</td></tr>
</table>

Usage example

```
//Added a variable to acquire one profile data of profile No.1 from batch No.1.
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("SPEC_BATCHPROFILE01","ProfileCount=1,BatchNo=1,ProfileNo=100");

Retrieve // profile data
object[] retValue = var.Value as object[];

object[] profileInfo = retValue[0] as object[]; // Profile info

byte? ProfileNum = profileInfo [0] as byte?; // Number of profiles
```

```
byte? Luminication = profileInfo [1] as byte?; // brightness information
short? ProfileDataCount profileInfo [2] as short?; // Number of profile data points
int? XPos = profileInfo [3] as int?; // X-coordinate
int? XInterval = profileInfo [4] as int?; // X spacing
uint? CurrentProfileNo = profileInfo [5] as uint?; // Latest profile number
uint? OldestProfileNo = profileInfo [6] as uint?; // Oldest profile number
uint? GetProfileCount = profileInfo [7] as uint?; // Number of profiles in the oldest batch
uint? GetOldestProfNo = profileInfo [8] as uint?; //oldest profile number read
uint? GetBatchNo = profileInfo [9] as uint?; // Batch number read
uint? GetProfileNo = profileInfo [10] as uint?; // Number of profiles in batches read
uint? GetBatchProfileNo = profileInfo [11] as uint?; // Which profile is
byte? GetProfileCount = profileInfo [12] as uint?; //oldest profile number read
byte? isBatchMesure = profileInfo [13] as byte?; //Whether the batch test has been completed

int?[] profileData = retValue[1] as int[]?; // Profile data
```

2.3.1.8. COMMITED_BATCHPROFILE<??>

After the batch is confirmed, the profile data is acquired from the latest. Retrieves the specified number of profile data by specifying the number of profile data to retrieve with the option.

The data to be acquired is the same as 2.3.1.6, so please refer to it.

<table>
<tr><th>Option</th><th>Required</th><th>Description</th><th>Value Range</th></tr>
<tr><td>TargetBank = [=
<acquisition plane>]</td><td>--</td><td>Specifies whether to acquire from an active or inactive plane.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>Active plane</td></tr><tr><td>1</td><td>Inactive plane</td></tr></table>Default: 0</td><td>0-1</td></tr>
<tr><td>ProfileCount
[=<number of profiles to retrieve>]</td><td>✓</td><td>Specify the number of profiles to read.</td><td>1-255</td></tr>
<tr><td>Erase = [<whether to delete profile data>]</td><td>--</td><td>Specify whether to erase the read profile and the previous profile.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>Without deleting</td></tr><tr><td>1</td><td>To delete</td></tr></table></td><td>0-1</td></tr>
</table>

Usage example

```
//add a variable to get one profile data
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("COMMITED_BATCHPROFILE01","Prof
ileCount=1");

// Retrieve profile data
object[] retValue = var.Value as object[];
```

```
object[] profileInfo = retValue[0] as object[]; // Profile info

byte? ProfileNum = profileInfo [0] as byte?; // Number of profiles
byte? Luminication = profileInfo [1] as byte?; // brightness information
short? ProfileDataCount profileInfo [2] as short?; // Number of profile data points
int? XPos = profileInfo [3] as int?; // X-coordinate
int? XInterval = profileInfo [4] as int?; // X spacing
uint? CurrentProfileNo = profileInfo [5] as uint?; // Latest profile number
uint? OldestProfileNo = profileInfo [6] as uint?; // Oldest profile number
uint? GetProfileCount = profileInfo [7] as uint?; // Number of profiles in the oldest batch
uint? GetOldestProfNo = profileInfo [8] as uint?; //oldest profile number read
uint? GetBatchNo = profileInfo [9] as uint?; // Batch number read
uint? GetProfileNo = profileInfo [10] as uint?; // Number of profiles in batches read
uint? GetBatchProfileNo = profileInfo [11] as uint?; // Which profile is
byte? GetProfileCount = profileInfo [12] as uint?; //oldest profile number read
byte? isBatchMesure = profileInfo [13] as byte?; //Whether the batch test has been completed

int?[] profileData = retValue[1] as int[]?; // Profile data
```

2.3.1.9. CURRENTONLY_BATCHPROFILE<??>

Retrieves only the latest profile data.

The data to be acquired is the same as 2.3.1.6, so please refer to it.

<table>
<tr><th>Option</th><th>Required</th><th>Description</th><th>Value Range</th></tr>
<tr><td>TargetBank = [=
<acquisition plane>]</td><td>--</td><td>Specifies whether to acquire from an active or inactive plane.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>Active plane</td></tr><tr><td>1</td><td>Inactive plane</td></tr></table>Default: 0</td><td>0-1</td></tr>
<tr><td>Erase = [<whether to delete profile data>]</td><td>--</td><td>Specify whether to erase the read profile and the previous profile.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>Without deleting</td></tr><tr><td>1</td><td>To delete</td></tr></table></td><td>0-1</td></tr>
</table>

Usage example

```
//add a variable to get one profile data
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("CURRENTONLY_BATCHPROFILE01",
"");

// Retrieve profile data
object[] retValue = var.Value as object[];
```

```
object[] profileInfo = retValue[0] as object[]; // Profile info

byte? ProfileNum = profileInfo [0] as byte?; // Number of profiles
byte? Luminication = profileInfo [1] as byte?; // brightness information
short? ProfileDataCount profileInfo [2] as short?; // Number of profile data points
int? XPos = profileInfo [3] as int?; // X-coordinate
int? XInterval = profileInfo [4] as int?; // X spacing
uint? CurrentProfileNo = profileInfo [5] as uint?; // Latest profile number
uint? OldestProfileNo = profileInfo [6] as uint?; // Oldest profile number
uint? GetProfileCount = profileInfo [7] as uint?; // Number of profiles in the oldest batch
uint? GetOldestProfNo = profileInfo [8] as uint?; //oldest profile number read
uint? GetBatchNo = profileInfo [9] as uint?; // Batch number read
uint? GetProfileNo = profileInfo [10] as uint?; // Number of profiles in batches read
uint? GetBatchProfileNo = profileInfo [11] as uint?; // Which profile is
byte? GetProfileCount = profileInfo [12] as uint?; //oldest profile number read
byte? isBatchMesure = profileInfo [13] as byte?; //Whether the batch test has been completed

int?[] profileData = retValue[1] as int[]?; // Profile data
```

2.3.1.10. CURRENT_BATCHPROFILE_SIMPLE<??>

Retrieves profile data from the latest. Retrieves the specified number of profile data by specifying the number of profile data to retrieve with the option.

The data to be retrieved is shown below.

Type	Description		
VT_ARRAY | VT_VARIANT			
0	VT_ARRAY | VT_VARIANT		
	0.0	VT_UI1	Gets the number of profiles stored in one unit of data.
	0.1	VT_UI1	Gets ON/OFF of brightness.
	0.2	VT_UI2	Gets the number of data points in one profile.
	0.3	VT_I4	Gets the X coordinate of the first point.
	0.4	VT_I4	X spacing of data points
	0.5	VT_I4	Gets the latest profile number at the time of acquisition.
	0.6	VT_UI4	Acquires the oldest profile number held by the controller.
	0.7	VT_UI4	Retrieves the number of profiles in the oldest batch held by the controller.
	0.8	VT_UI4	Acquires the oldest profile number that has been read.
	0.9	VT_UI4	Acquires the batch number read this time.
	0.10	VT_UI4	Gets the number of profiles in the batch read this time.
	0.11	VT_UI4	Gets the number of profiles in the read batch.
	0.12	VT_UI1	Gets the number of profiles read this time.
	0.13	VT_UI1	Acquires whether or not the latest batch measurement has been completed.

1. N	VT_ARRAY|VT_I4	The header information of the read profile data is acquired.
2. N	VT_ARRAY|VT_I4	The height information of the read profile data is acquired.
3. N	VT_ARRAY|VT_I4	Brightness information of the read profile data is acquired.

<table>
<tr><th>Option</th><th>Required</th><th>Description</th><th>Value Range</th></tr>
<tr><td>TargetBank = [=
<acquisition plane>]</td><td>--</td><td>Specifies whether to acquire from an active or inactive plane.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>Active plane</td></tr><tr><td>1</td><td>Inactive plane</td></tr></table>Default: 0</td><td>0-1</td></tr>
<tr><td>ProfileCount
[=<number of profiles to retrieve>]</td><td>✓</td><td>Specify the number of profiles to read.</td><td>1-255</td></tr>
<tr><td>Erase = [<whether to delete profile data>]</td><td>--</td><td>Specify whether to erase the read profile and the previous profile.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>Without deleting</td></tr><tr><td>1</td><td>To delete</td></tr></table></td><td>0-1</td></tr>
</table>

Usage example

```
// Added a variable that gets one profile count from the profile No10
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("CURRENT_PROFILE_SIMPLE01","Prof
ileCount=1");

// Retrieve profile data
object[] retValue = var.Value as object[];

object[] profileInfo = retValue[0] as object[]; // Profile info

byte? ProfileNum = profileInfo [0] as byte?; // Number of profiles
byte? Luminication = profileInfo [1] as byte?; // brightness information
short? ProfileDataCount profileInfo [2] as short?; // Number of profile data points
int? XPos = profileInfo [3] as int?; // X-coordinate
int? XInterval = profileInfo [4] as int?; // X spacing
uint? CurrentProfileNo = profileInfo [5] as uint?; // Latest profile number
uint? OldestProfileNo = profileInfo [6] as uint?; // Oldest profile number
uint? GetProfileCount = profileInfo [7] as uint?; // Number of profiles in the oldest batch
uint? GetOldestProfNo = profileInfo [8] as uint?; //oldest profile number read
uint? GetBatchNo = profileInfo [9] as uint?; // Batch number read
```

```
uint? GetProfileNo = profileInfo [10] as uint?; // Number of profiles in batches read
uint? GetBatchProfileNo = profileInfo [11] as uint?; // Which profile is
byte? GetProfileCount = profileInfo [12] as uint?; //oldest profile number read
byte? isBatchMesure = profileInfo [13] as byte?; //Whether the batch test has been completed

int?[] profileData = retValue[1] as int[]?; // profile header data
int?[] profileData = retValue[2] as int[]?; // Profile height data
int?[] profileData = retValue[3] as int[]?; // Profile brightness data
```

2.3.1.11. SPEC_BATCHPROFILE_SIMPLE<??>

Retrieves profile data from a location that you specify. Specify the batch No. and profile No. to be acquired optionally, and obtain the specified number of profile data from the specified location.

The data to be acquired is the same as 2.3.1.10, so please refer to it.

Option	Required	Description	Value Range
TargetBank = [= <acquisition plane>]	--	Specifies whether to acquire from an active or inactive plane. Set value: 0 = Active plane; 1 = Inactive plane Default: 0	0-1
BatchNo = [= <Batch No>]	✓	Specifies the batch number for which profile data is to be retrieved.	0~4294967295
ProfileNo = [= <profile number to be acquired>]	✓	When 2 is specified in the profile position specification method, specify the profile number to be acquired.	0~4294967295
ProfileCount [=<number of profiles to retrieve>]	✓	Specify the number of profiles to read.	1-255
Erase = [<whether to delete profile data>]	--	Specify whether to erase the read profile and the previous profile. Set value: 0 = Without deleting; 1 = To delete	0-1

Usage example

```
//Added a variable to acquire one profile data of profile No.1 from batch No.1.
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("SPEC_BATCHPROFILE_SIMPLE01","P
rofileCount=1,BatchNo=1,ProfileNo=100");

// Retrieve profile data
object[] retValue = var.Value as object[];
```

```
object[] profileInfo = retValue[0] as object[]; // Profile info

byte? ProfileNum = profileInfo [0] as byte?; // Number of profiles
byte? Luminication = profileInfo [1] as byte?; // brightness information
short? ProfileDataCount profileInfo [2] as short?; // Number of profile data points
int? XPos = profileInfo [3] as int?; // X-coordinate
int? XInterval = profileInfo [4] as int?; // X spacing
uint? CurrentProfileNo = profileInfo [5] as uint?; // Latest profile number
uint? OldestProfileNo = profileInfo [6] as uint?; // Oldest profile number
uint? GetProfileCount = profileInfo [7] as uint?; // Number of profiles in the oldest batch
uint? GetOldestProfNo = profileInfo [8] as uint?; //oldest profile number read
uint? GetBatchNo = profileInfo [9] as uint?; // Batch number read
uint? GetProfileNo = profileInfo [10] as uint?; // Number of profiles in batches read
uint? GetBatchProfileNo = profileInfo [11] as uint?; // Which profile is
byte? GetProfileCount = profileInfo [12] as uint?; //oldest profile number read
byte? isBatchMesure = profileInfo [13] as byte?; //Whether the batch test has been completed

int?[] profileData = retValue[1] as int[]?; // profile header data
int?[] profileData = retValue[2] as int[]?; // Profile height data
int?[] profileData = retValue[3] as int[]?; // Profile brightness data
```

2.3.1.12. COMMITED_BATCHPROFILE_SIMPLE<??>

After the batch is confirmed, the profile data is acquired from the latest. Retrieves the specified number of profile data by specifying the number of profile data to retrieve with the option.

The data to be acquired is the same as 2.3.1.10, so please refer to it.

<table>
<tr><th>Option</th><th>Required</th><th>Description</th><th>Value Range</th></tr>
<tr><td>TargetBank = [=
<acquisition plane>]</td><td>--</td><td>Specifies whether to acquire from an active or inactive plane.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>Active plane</td></tr><tr><td>1</td><td>Inactive plane</td></tr></table>Default: 0</td><td>0-1</td></tr>
<tr><td>ProfileCount
[=<number of profiles to retrieve>]</td><td>✓</td><td>Specify the number of profiles to read.</td><td>1-255</td></tr>
<tr><td>Erase = [<whether to delete profile data>]</td><td>--</td><td>Specify whether to erase the read profile and the previous profile.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>Without deleting</td></tr><tr><td>1</td><td>To delete</td></tr></table></td><td>0-1</td></tr>
</table>

Usage example

```
//add a variable to get one profile data
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("COMMITED_BATCHPROFILE_SIMPLE01","ProfileCount=1");
```

```
// Retrieve profile data
object[] retValue = var.Value as object[];

object[] profileInfo = retValue[0] as object[]; // Profile info

byte? ProfileNum = profileInfo [0] as byte?; // Number of profiles
byte? Luminication = profileInfo [1] as byte?; // brightness information
short? ProfileDataCount profileInfo [2] as short?; // Number of profile data points
int? XPos = profileInfo [3] as int?; // X-coordinate
int? XInterval = profileInfo [4] as int?; // X spacing
uint? CurrentProfileNo = profileInfo [5] as uint?; // Latest profile number
uint? OldestProfileNo = profileInfo [6] as uint?; // Oldest profile number
uint? GetProfileCount = profileInfo [7] as uint?; // Number of profiles in the oldest batch
uint? GetOldestProfNo = profileInfo [8] as uint?; //oldest profile number read
uint? GetBatchNo = profileInfo [9] as uint?; // Batch number read
uint? GetProfileNo = profileInfo [10] as uint?; // Number of profiles in batches read
uint? GetBatchProfileNo = profileInfo [11] as uint?; // Which profile is
byte? GetProfileCount = profileInfo [12] as uint?; //oldest profile number read
byte? isBatchMesure = profileInfo [13] as byte?; //Whether the batch test has been completed

int?[] profileData = retValue[1] as int[]?; // profile header data
int?[] profileData = retValue[2] as int[]?; // Profile height data
int?[] profileData = retValue[3] as int[]?; // Profile brightness data
```

2.3.1.13. CURRENTONLY_BATCHPROFILE_SIMPLE<??>

Retrieves only the latest profile data.

The data to be acquired is the same as 2.3.1.10, so please refer to it.

<table>
<tr><th>Option</th><th>Required</th><th>Description</th><th>Value Range</th></tr>
<tr><td>TargetBank = [=
<acquisition plane>]</td><td>--</td><td>Specifies whether to acquire from an active or inactive plane.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>Active plane</td></tr><tr><td>1</td><td>Inactive plane</td></tr></table>Default: 0</td><td>0-1</td></tr>
<tr><td>Erase = [<whether to delete profile data>]</td><td>--</td><td>Specify whether to erase the read profile and the previous profile.
<table><tr><th>Set value</th><th>Introduction</th></tr><tr><td>0</td><td>Without deleting</td></tr><tr><td>1</td><td>To delete</td></tr></table></td><td>0-1</td></tr>
</table>

Usage example

```
//add a variable to get one profile data
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("CURRENTONLY_BATCHPROFILE_SIMPLE01","");
```

```
// Retrieve profile data
object[] retValue = var.Value as object[];

object[] profileInfo = retValue[0] as object[]; // Profile info

byte? ProfileNum = profileInfo [0] as byte?; // Number of profiles
byte? Luminication = profileInfo [1] as byte?; // brightness information
short? ProfileDataCount profileInfo [2] as short?; // Number of profile data points
int? XPos = profileInfo [3] as int?; // X-coordinate
int? XInterval = profileInfo [4] as int?; // X spacing
uint? CurrentProfileNo = profileInfo [5] as uint?; // Latest profile number
uint? OldestProfileNo = profileInfo [6] as uint?; // Oldest profile number
uint? GetProfileCount = profileInfo [7] as uint?; // Number of profiles in the oldest batch
uint? GetOldestProfNo = profileInfo [8] as uint?; //oldest profile number read
uint? GetBatchNo = profileInfo [9] as uint?; // Batch number read
uint? GetProfileNo = profileInfo [10] as uint?; // Number of profiles in batches read
uint? GetBatchProfileNo = profileInfo [11] as uint?; // Which profile is
byte? GetProfileCount = profileInfo [12] as uint?; //oldest profile number read
byte? isBatchMesure = profileInfo [13] as byte?; //Whether the batch test has been completed

int?[] profileData = retValue[1] as int[]?; // profile header data
int?[] profileData = retValue[2] as int[]?; // Profile height data
int?[] profileData = retValue[3] as int[]?; // Profile brightness data
```

2.4. Event list

You can receive controller error notifications and status changes as OnMessage events.

XX of the message number is set to XX = DeviceID; if DeviceID option specified during AddController is set to 1, the message number is set to 1.

The content of the message can be obtained with Value property.

Message number	Description	See Also
XX(0~127)	Issue the profile information as a message.	P.43
1XX(256~383)	Issues a messagewhen the profile count specified by HSDComm option during StartHighSpeedDataCommunication or AddController is received.	P.43
2XX(512~639)	Issue messaging when the profile count specified by StartHighSpeedData CommunicationSimpleArray is received.	P.43
3XX(768~895)	Issue the end message of high-speed data communication. This messaging is issued when HSDCommEndMess option is enabled during AddController.	P.44

2.4.1. Profile information message

Issues a messagewhen the profile count specified by HSDComm option during StartHighSpeedDataComm unication or AddController is received. The contents of the message are shown below.

Data Type

Item	Type Description		
Message Overview	VT_ARRAY | VT_VARIANT		
	0.0	VT_UI1	Gets the number of profiles stored in one unit of data.
	0.1	VT_UI1	Gets ON/OFF of brightness.
	0.2	VT_UI2	Gets the number of data points in one profile.
	0.3	VT_I4	Gets the X coordinate of the first point.
	0.4	VT_I4	Gets the X distance between data points.

2.4.2. Profile data message

Issue the profile information. This event is issued at the same time when profile data is received. The contents of the message are shown below. The messaging number is 0x100(256) + DeviceId. If DeviceId specified in 2.2.1.1 is 10, 0x10A(266) is the message number.

Data Type

Item	Type Description		
Message Overview	VT_ARRAY | VT_VARIANT		
	N	VT_ARRAY|VT_I4	Get profile data. The acquired profile data is stored in order.

2.4.3. SimpleArray Profile Information Messages

Issue the profile that is returned during StartHighSpeedDataCommunicationSimpleArray commandexecution. This event is issued at the same time when profile data is received. The contents of the message are shown below.

Data Type

Item	Type Description		
Message Overview	0.N	VT_ARRAY|VT_I4	Gets header information for each profile data.
	1.N	VT_ARRAY|VT_UI2	Retrieves the height information of the read profile data.
	2.N	VT_ARRAY|VT_UI2	Acquires the brightness data of the read profile data.

2.4.4. High-speed data communication end message

Issued as a message when high-speed data communication ends. This event is issued only when HSDCommEndMess option is enabled during AddController. The contents of the message are shown below.

Data Type

<table>
<tr><th>Item</th><th>Type Description</th></tr>
<tr><td>Message Overview</td><td>Acquires the value of the end event of high-speed data communication.
The values and their contents are shown below.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>1</td><td>High-speed data communication was terminated by StopHighSpeedDataCommunication command.</td></tr>
<tr><td>2</td><td>High-speed data communication is terminated because the setting has been changed.</td></tr>
<tr><td>4</td><td>High-speed data communication terminated because the execution program was switched.</td></tr>
<tr><td>8</td><td>High-speed data communication was forcibly stopped.</td></tr>
<tr><td>256</td><td>High-speed data communication is terminated by memory clear.</td></tr>
</table></td></tr>
</table>

3. Programming by LJ-X8000A providers

With LJ-X8000A providers, you can connect LJ-X8000A to the client computer as follows:

- Creating a CaoEngine
- Creating a CaoWorkspace
- Creating a CaoController

After you connect to LJ-X8000A, you can access CaoController by using Execute method.

3.1. Sample Programming to Retrieve Profile Data

This section shows how to retrieve profile data from LJ-X8000A. Table 3-1 Sample program requirements describes the requirements of the sample program, and Fig. 3-1 describes the flow of the sample program.

Table 3-1 Sample program requirements

Requirements	Description
Host	Connect with a TCP/IP
	The destination IP address is 192.168.1.2.
	The destination port number is 24691.
Process Description	Retrieving Profile Data from a LJ-X8000A
	Dividing the contents of acquired profile data by item

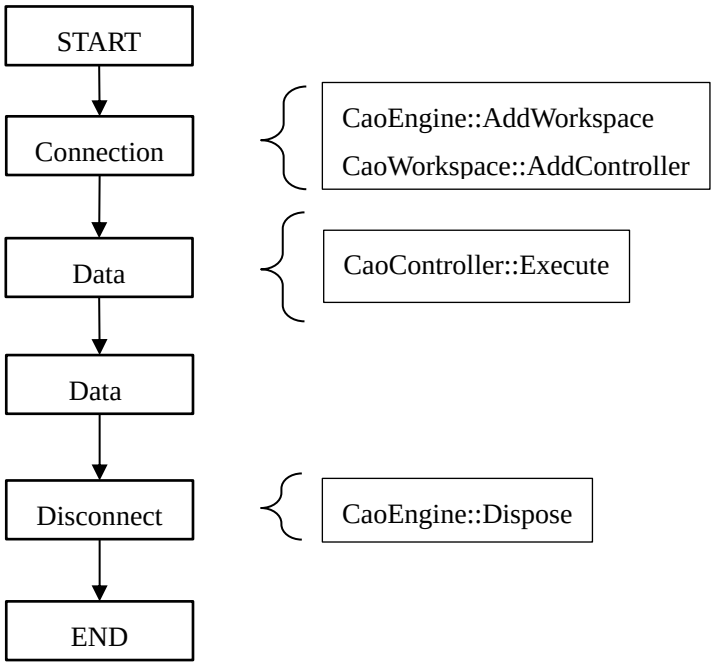


Fig. 3-1 Flow of profile data acquisition

Specific codes are given in the following sections.

3.1.1. Sample program

The following is an overview of the sample program.

Sample **Sample.cs**

```
// Object
private ORiN2.ManagedCAO.CCaoEngine m_caoEngine = null;
private ORiN2.ManagedCAO.CCaoWorkspace m_caoWorkspace = null;
private ORiN2.ManagedCAO.CCaoController m_caoController = null;

public void Main()
{
        // Connection
        this.Connect();

        // Retrieving Profile Data
        object[] prm = new object[] {0, 1, 1, 5, 1};
        object[] retVarue = this.m_caoController.Execute("GetProfile", prm) as object[];

        object[] deviceparm = retVarue[0] as object[];
        int?[] profileDataArray = retVarue[1] as int?[];

        byte? profCount = deviceparm[0] as byte?; //Number of profiles per unit
        byte? luminanceOutput = deviceparm[1] as byte?;//Brightness ON/OFF
        ushort? profDataCount = deviceparm[2] as ushort?; //Number of data points in 1 profile
        int? lXStart = deviceparm[3] as int?; // X-coordinate of 1st point
        int? lXPitch = deviceparm[4] as int?; // X spacing of data points

        // Disconnect
        this.Disconnect();
}

// Connection method
private void Connect()
{
    // Generate CaoEngine object
    this.m_caoEngine = new ORiN2.ManagedCAO.CCaoEngine();
    // Generate CaoWorkspace object
    this.m_caoWorkspace = this.m_caoEngine.AddWorkspace("NewWrks", "");
    // Generate CaoController object
    this.m_caoController = this.m_caoWorkspace.AddController("LJX8000A",
                                    "CaoProv.KEYENCE.LJ-X8000",
                                    "",
                                    "Conn=ETH:127.0.0.1,DeviceID=1,Head=1, Timeout=1000");
}

// Disconnect method
private void Disconnect()
{
    This.m_caoEngine.Dispose();
    This.m_caoEngine = null;
}
```

3.1.1.1. Connection

To connect to LJ-X8000A, perform the following steps:

(1) Prepare a variable to hold the object. The objects required to connect to the controller are CaoEngine object, CaoWorkspace object, and CaoController object. CaoWorkpace object does not need to have a variable to obtain CaoController object from CaoWorkspaces. . The following is a code example for C#.

```
// Variables for CaoEngine
private ORiN2.ManagedCAO.CCaoEngine m_caoEngine = null;
// Variables for CaoWorkspace
private ORiN2.ManagedCAO.CCaoWorkspace m_caoWorkspace = null;
// Variables for CaoController
private ORiN2.ManagedCAO.CCaoController m_caoController = null;
```

(2) Creates a CaoEngine object. CaoEngine object is generated using the New keyword.

```
// Generate CaoEngine object
this.m_caoEngine = new ORiN2.ManagedCAO.CCaoEngine();
```

(3) Gets or generates a CaoWorkspace object. When you create a CaoEngine object, it defaults to one CaoWorkspaces object and one object. The following is a sample code/default CaoWorkspace for creating a new CaoWorkspace.

```
// Generate CaoWorkspace object
this.m_caoWorkspace = this.m_caoEngine.AddWorkspace("NewWrks", "");
```

(4) Create a CaoController object. To create a CaoController object, set the provider name to use and the parameters to use. For LJ-X8000A providers, optionally specify the ETH connection, device number (optional), and header number. The following is a code example:

```
// Generate CaoController object
this.m_caoController = this.m_caoWorkspace.AddController("LJX8000A",
                                    "CaoProv.KEYENCE.LJ-X8000",
                                    "",
                                    "Conn=ETH:127.0.0.1,DeviceID=1,Head=1, Timeout=1000");
```

3.1.1.2. Retrieving Profile Data

Use Execute command to acquire profile data. For details of each parameter, refer to 2.2.2.4.15. This time, the first five profile information are set to be acquired. Each data is stored in a separate variable after retrieval.

```
// Parameter specification
object[] prm = new object[] {0, 1, 1, 5, 1};
// Retrieving Profile Data
object[] retVarue = this.m_caoController.Execute("GetProfile", prm) as object[];

object[] Profparm = retVarue[0] as object[]; // Profile setting items
int?[] profileDataArray = retVarue[1] as int?[]; //5 minutes profile data
```

```
byte? profCount = Profparm [0] as byte?; //Number of profiles per unit
byte? luminanceOutput = Profparm [1] as byte?;//Brightness ON/OFF
ushort? profDataCount = Profparm [2] as ushort?; //Number of data points in one profile
int? lXStart = Profparm [3] as int?; // X-coordinate of 1st point
int? lXPitch = Profparm [4] as int?; // X spacing of data points
```

3.1.1.3. Disconnect

To disconnect from the controller, you can erase the generated objects and delete the objects that you want to erase from the collection class that manages the objects. However, you do not need to explicitly remove it if you use ORiN.ManagedCAO. The following is a code example:

```
// Remove all objects from CaoEngine
this.m_caoEngine.Dispose();
// Clear CaoEngine
this.m_caoEngine = null;
```

4. LJ-X8000A Provider Error Codes

For information about common ORiN2 errors, see the Error Codes section of ORiN2 Programming Guide.

Table 4-1 Unique Error Codes

Error Number	Description
0x8010xxxx	Doing so results in an error from the library.

This provider also masks the error code of the library with "0x8010****" and returns it.

For details of error codes, see "LJ-V8000A series communication library".

Appendix A. API correspondence table

This is the correspondence between the commands used in LJX-8000A and the APIs in LJX8IF.dll.

LJ-X8000A Provider Commands	APIs for LJX8IF.dll
GetError	LJX8IF_GetError
ClearError	LJX8IF_ClearError
TrgErrorReset	LJX8IF_TrgErrorReset
GetTriggerAndPulseCount	LJX8IF_GetTriggerAndPulseCount
GetHeadTemperture	LJX8IF_GetHeadTemperature
GetSerialNumber	LJX8IF_GetSerialNumber
GetAttentionStatus	LJX8IF_GetAttentionStatus
GetLedLightImage	LJX8IF_GetLedLightImage
Trigger	LJX8IF_Trigger
StartMesure	LJX8IF_StartMeasure
STOPMESURE	LJX8IF_STOPMEASURE
ClearMemory	LJX8IF_ClearMemory
GetActiveProgram	LJX8IF_GetActiveProgram
ChangeActiveProgram	LJX8IF_ChangeActiveProgram
GetProfile	LJX8IF_GetProfile
GetBatchProfile	LJX8IF_GetBatchProfile
GetbatchSimpleArray	LJX8IF_GetBatchSimpleArray
StartHighSpeedDataCommunication	LJX8IF_InitializeHighSpeedDataCommunication LJX8IF_PreStartHighSpeedDataCommunication LJX8IF_StartHighSpeedDataCommunication
StartHighSpeedDataCommunicationSimpleArray	LJX8IF_InitializeHighSpeedDataCommunicationSimpleArray LJX8IF_PreStartHighSpeedDataCommunication LJX8IF_StartHighSpeedDataCommunication
StopHighSpeedDataCommunication	LJX8IF_StopHighSpeedDataCommunication LJX8IF_FinalizeHighSpeedDataCommunication