SOFIX CORPORATION

SOFIXCAN ΩEye Provider

User's guide

Version 1.0.0

July 17, 2024

Remarks: The acquisition of SOFIXCAN ΩEye for Windows data using this provider requires the purchase of the ORiN option (paid option) for ΩEye.

※For details on ORiN options and other ΩEye-related information, please contact the following

SOFIXCAN ΩEye Official : sofixcan-omega-eye@sofix.co.jp

【 Revision history 】

Version	Date	Content
1.0.0	2024-07-17	First edition.

【 Supported model 】

Model	Version	Notes
SOFIXCAN ΩEye for Windows	3.1E	Only Rev. 3.1E or later versions are supported.

【 Operation check model 】

Model	Version	Notes
SOFIXCAN ΩEye for Windows	3.1E	Other versions are not guaranteed.

CONTENTS

1. Introduction

This document is a user's guide for providers accessing data to SOFIX's SOFIXCAN ΩEye device. Fig. 1-1 shows the overall configuration of the Provider and the device. This provider is hereinafter referred to as the ΩEye Provider.

(※ΩEye provider is supported only for ΩEye Windows version. ΩEye hardware version is not supported.)

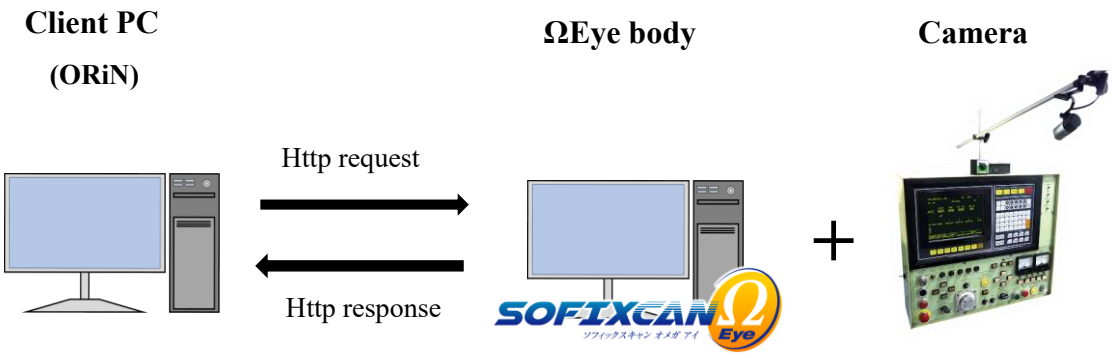


Fig. 1-1 Configuration

The correspondence between the Provider and the device is shown in Fig. 1-2.

(* This is an example. Not all of them are shown.)

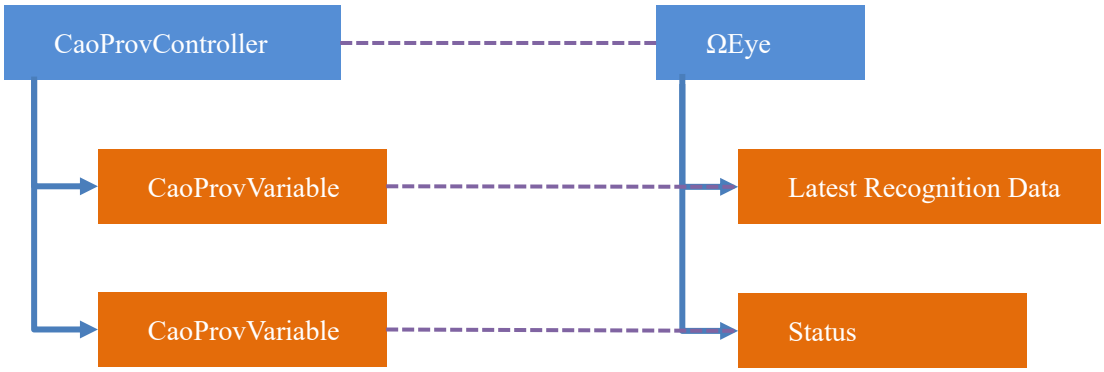


Fig. 1-2 Correspondence diagram between provider configuration and device information

2. Environment setup for application development

2.1. Connecting ΩEye to Client PC

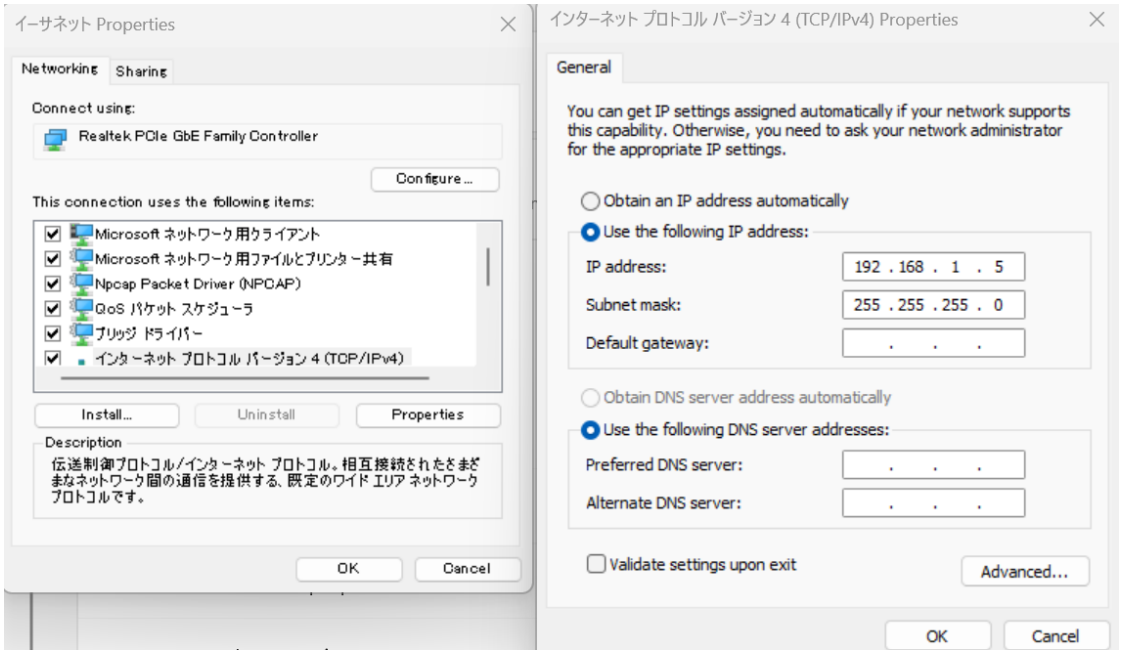
ΩEye and ΩEye provider communicate using ΩEye WebAPI. ΩEye WebAPI is a function for viewing and acquiring recognition data created by ΩEye from external systems via HTTP communication.

Follow the procedure below to connect ΩEye to the client PC.

1 Enable the WebAPI function according to Chapters 7, 8 and 9 of the SOFIXCAN Ω Eye for Windows Installation Manual.

2 Check or change the IPv4 settings of the PC on which ΩEye is installed (ΩEye PC).

3 Set the IP address of the client PC to one that can communicate with the IP address set in the ΩEye PC.



※ ΩEye providers cannot select network adapters or specify whether to use proxies. The provider automatically determines the setting according to the client PC's configuration. If an unexpected network adapter is used or a proxy is used and communication is not possible, review the client PC settings.

2.2. Setting up the PC development environment

2.2.1. Manual Installation of ΩEye Provider

To use the ΩEye provider, you must register according to Table 2-1. RegistAsm.bat and UnregistAsm.bat are in the DotNet¥BAT folder under the folder where ORiN2SDK is installed.

Table 2-1 ΩEye Provider

File name	CaoProvSOFIXSOFIXCANOmegaEyeV3.exe
ProgID	CaoProv.SOFIX.SOFIXCAN.OmegaEye.V3
Registry registration[1]	RegistAsm.bat CaoProvSOFIXSOFIXCANOmegaEyeV3.exe
Blotting out of registry registration	UnregistAsm.bat CaoProvSOFIXSOFIXCANOmegaEyeV3.exe

[1] If installed with ORiN SDK, there is no need to manually regist / delete.

3. Command Reference

3.1. List of Methods / Properties

Table 3-1 List of Methods/Properties

Category	Method / Property[2]		Function	Reference
CaoWorkspace				
	AddController	M	Connects to controller.	P.8
CaoController				
	GetVariableNames	M	Gets a list of connectable variable names.	P.9
	Variables	P	Gets the collection of variables held by the controller.	P.10
	AddVariable	M	Adds a variable object.	P.10
CaoVariable				
	Value	P	Gets value.	P.10

3.2. Methods and Properties

3.2.1. CaoWorkspace class

3.2.1.1. AddController method

Adds a controller object to CaoWorkspace. The ΩEye provider does not make a connection when the AddController method is executed. When acquiring the value of the Value property of the CaoVariable class, the passed parameter is referenced to access the corresponding ΩEye web server. The specifications of the AddController method are shown below.

Format

```
AddController
(
        "<Controller Name>",                    // Controller name (arbitrary)
        "CaoProv.SOFIX.SOFIXCAN.OmegaEye.V3",   // Provider name (fixed)
        "<Machine Name>",                       // Provider execution machine name (unused)
        "<Option>"                              // Option string
)
```

Option

[2] M : Method, P : Property, E : Event, respectively.

The following is a list of options to be specified in the option string. The option string is a string consisting of the following options separated by commas (,).

Option	Required	Description	Value range	Default value
Uri	✓	Specify the URL to access ΩEye, the connection destination. For the access URL, refer to 3.2.1.1.1.	-	-
Timeout	-	Specify the response waiting time.	0 - 600000	2000

Examples of use (C#)

```
// Engine Object
ORiN2.ManagedCAO.CCaoEngine engine = new ORiN2.ManagedCAO.CCaoEngine();
// Workspace Object
ORiN2.ManagedCAO.CCaoWorkspace workspace = engine.AddWorkspace("NewWrks", "");
// Controller Object
ORiN2.ManagedCAO.CCaoController controller= workspace.AddController("OmegaEye",
                                                    "CaoProv.SOFIX.SOFIXCAN.OmegaEye.V3",
                                                    "",
                                                    "Uri = http:/192.168.0.1/OMEGA-EYE-API ");
```

3.2.1.1.1. Access URL

"OMEGA-EYE-API" must be specified after the IP address of ΩEye. For details, please refer to the table below.

Table 3-2 Access URL

OS	Access URL example
Windows Version	"http:/192.168.0.1OMEGA-EYE-API"

3.2.2. CaoController Class

3.2.2.1. GetVariableNames Method

Gets a list of connectable variable names. For available variables, see 3.3 Variable List.

Format

```
GetVariableNames
(
        "<Option>"                          // Option string (unused)
)
```

Example of use (C#)

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

3.2.2.2. Variables Property

Gets a collection of variables held by the controller.

Example of use (C#)

```
// Get variable collection
ORiN2.ManagedCAO.CCaoVariables variables = controller.Variables;
// Get variable
ORiN2.ManagedCAO.CCaoVariable variable = variables[0];
```

3.2.2.3. AddVariable Method

Adds variable objects to CaoController. Only those variable names shown in 3.3.2 may be used.

The specifications of AddVariable are shown below.

Format

```
AddVariable
(
        "<Variable Name>",          // Variable name
        "<Option>"                  // Option string (optional)
)
```

3.2.3. CaoVariable Class

3.2.3.1. Value Property

Connects to ΩEye and get data. The behavior differs depending on the variable name. For details, see 3.3. Variable List.

3.3. Variable List

Defines a list of variables that can be used in each class. The variables point to objects of the CaoVariable class.

3.3.1. System variables and user variables

The provider has two types of variables: system variables and user variables.

System variable

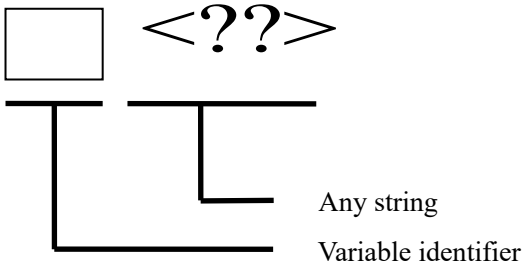
System variable is used to access the only information in the object that holds the variable. System variables are often static data. System variables have "@" at the beginning of their names.

e.g.) Provider version, device manufacturer, recognition processing status of recognition application.

User variable

User variable is a variable that allows the user to specify what information is to be accessed. User variables, by their nature, can have arbitrary strings added after the variable identifier in order to register multiple variables (useful, for example, when only options need to be changed). The format for adding arbitrary strings to variable names is shown below.

Common specification format for multiple variables



3.3.2. CaoController Class variables

Variable Name	Description	Value		Reference
		get	put	
@MAKER_NAME	Gets maker name.	✓	-	P.12
@VERSION	Gets DLL version.	✓	-	P.12
@STATUS	Gets recognition application status.	✓	-	P.12
@SYSTEM_INFO	Gets the version information of the recognition application, etc.	✓	-	P.13
LAST_RESULT	Gets the latest recognition result data.	✓	-	P.14
RESULTS	Gets the specified number of recognition results backward from the most recent data.	✓	-	P.14

3.3.2.1. @MAKER_NAME

Gets maker name.

Data type

Type description	
VT_BSTR	Gets maker name.

Example of use (C#)

```
// Add variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@MAKER_NAME","");
// Get value
string value = var.Value as string;
```

3.3.2.2. @VERSION

Gets Provider version.

Data type

Type description	
VT_BSTR	Gets ΩEye Provider version. *.*.*

Example of use (C#)

```
// Add variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@VERSION","");
// Get value
string value = var.Value as string;
```

3.3.2.3. @STATUS

Gets the recognition processing status of the recognition application.

Data type

Type description	
VT_BSTR	Gets the recognition processing status of the recognition application. The meaning of the values you get is shown below. "active": The application is running and processing recognition. (Shooting method is "timer type".) "stopped": Recognition application is running but recognition process is stopped. (Shooting method is "timer type".) "inactive": Recognition application is not running. "camera_ready": The application is running with "external trigger" as the shooting method and waiting for a one-shot recognition request. "invalid_license" : ORiN option is not valid. The ORiN option of ΩEye is in an invalid state, and the option needs to be enabled. Please contact the following for details. ・SOFIXCAN ΩEye Official : sofixcan-omega-eye@sofix.co.jp

Example of use (C#)

```
// Add variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@STATUS","");
// Get value
string value = var.Value as string;
```

3.3.2.1. @SYSTEM_INFO

Gets the version information of the recognition application, etc.

Data type

Type description		
VT_ARRAY | VT_BSTR		
	0	Gets software version. *.*.*
	1	Gets the individual name set in the camera setting screen of the setting application.

Example of use (C#)

```
// Add variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@SYSTEM_INFO","");
// Get value
string[] values = var.Value as string[];
```

```
// Software version
var version = values[0];
// Individual name
var name = values[1];
```

3.3.2.2. LAST_RESULT<??>

Gets the latest recognition result data. Enter any string after LAST_RESULT to specify the variable name.

Option

Option	Required	Description	Value range	Default value
SettingId	-	Specify the profile number of the recognition result data. Specify 0 in the Standard version. See "Appendix. B. About Profiles" for more information on profiles.	0 - 9	0

※ Recognition result data is published under the directory for each profile number, such as "measurement_0", "measurement_1".

Data type

Type description	
VT_ARRAY | VT_BSTR	A single CSV line of recognition result data is divided by comma delimitation and stored as array data.

Example of use (C#)

```
// Add variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("LAST_RESULT1"," SettingId = 0");
// Get value
string[] values = var.Value as string[];

for (var i = 0; i < values.Length; i++)
{
    // Comma-separated data
    var splitData = values[i];
}
```

3.3.2.3. RESULTS<??>

Gets the specified number of recognition results backward from the most recent data. Enter any string after RESULTS to specify the variable name.

Option

Option	Required	Description	Value range	Default value
SettingId	-	Specify the profile number of the recognition result data. Specify 0 in the Standard version. See "Appendix. B. About Profiles" for more information on profiles.	0 - 9	0
Count	-	Specify the number of data you want to get.	1 - 2147483647	1

※ Recognition result data is published under the directory for each profile number, such as "measurement_0", "measurement_1".

Data type

Type description		
VT_ARRAY | VT_VARIANT		Gets a historical list of recognition result data. The first line is the latest data.
i	VT_ARRAY VT_BSTR	Divides a line of CSV of recognition result data by comma delimitation and stores CSV as a jag array, where each line is stored as array data.

※ **i**: The number of data history.

Example of use (C#)

```
// Add variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("RESULTS1", " SettingId = 0,Count = 1");

// Get value
object[] results = var.Value as object[];

for (int i = 0; i < results.Length; i++)
{
    // Recognition result data
    string[] resultData = results[i] as string[];

    for (int j = 0; j < resultData.Length; j++)
    {
        // Comma-separated data
        string splitData = resultData[j];
    }
}
```

4. Programming by ΩEye provider

The ΩEye provider can connect a client PC to the ΩEye by the following procedure.

- Create CaoEngine
- Create CaoWorkspace
- Create CaoController
- Create CaoVariable
- Execute getting Value property of CaoVariable

4.1. Sample programming to get the latest recognition result data

Here is a sample program to get the latest recognition data of ΩEye as an example. Table 4-1 and Fig. 4-1 describe the requirements and flow of the sample program, respectively.

Table 4-1 Requirements for the Sample Program

Requirement	Description
Access point	Connect with Http communication.
	IP address to connect to is 192.168.0.1.
	ΩEye is Standard version.
Process detail	Read the latest recognition results from ΩEye.

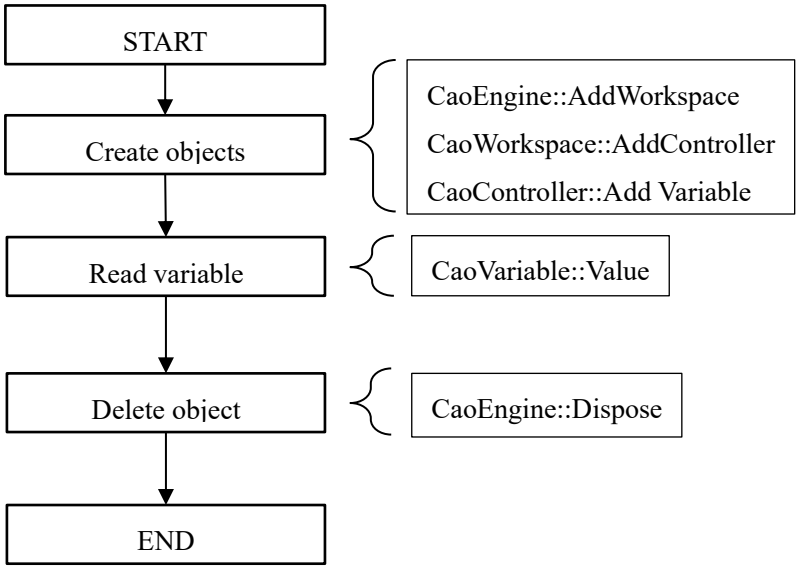


Fig. 4-1 Process Flow

The following sections show specific code.

4.1.1. Sample program

The following is an overview of the sample program.

Sample	Sample.cs

```
private ORiN2.ManagedCAO.CCaoEngine m_caoEngine = null;
private ORiN2.ManagedCAO.CCaoWorkspace m_caoWorkspace = null;
private ORiN2.ManagedCAO.CCaoController m_caoController = null;
private ORiN2.ManagedCAO.CCaoVariable m_varLastResult = null;

public void Main()
{
    // Create object
    this.CreateObject();

    // Read variable
    string[] values = this.m_varLastResult.Value as string[];

    for (var i = 0; i < values.Length; i++)
    {
        // Get comma-separated data
        var splitData = values[i];
    }

    // Delete object
    this.DeleteObject();
}

// Create object method
private void CreateObject ()
{
    // Create CaoEngine object
    this.m_caoEngine = new ORiN2.ManagedCAO.CCaoEngine();
    // Create CaoWorkspace object
    this.m_caoWorkspace = this.m_caoEngine.AddWorkspace("NewWrks", "");
    // Create CaoController object
    this.m_caoController = this.m_caoWorkspace.AddController("OmegaEye",
                                                             "CaoProv.SOFIX.SOFIXCAN.OmegaEye.V3",
                                                             "",
```

```
                                                    "Uri = http://192.168.0.1/OMEGA-EYE-API");
        // Create CaoVariable object
        this.m_varLastResult = this.m_caoController.AddVariable("LAST_RESULT1"," SettingId = 0");
}

    // Delete object method
private void DeleteObject ()
{
        this.m_caoEngine.Dispose();
        this.m_caoEngine = null;
}
```

4.1.1.1. Create objects

The following steps are taken as a preliminary step to get data.

(1) Prepare variables to hold objects.

The objects required to get data are a CaoEngine object, a CaoWorkspace object, and a CaoController object. A CaoWorkspace object does not require any variables when you get a CaoController object from CaoWorkspaces. A CaoVariable object is also required to access the variables. An example code in C# is shown below.

Example of use (C#)

```
// Variable for CaoEngine object
private ORiN2.ManagedCAO.CCaoEngine m_caoEngine = null;
// Variable for CaoWorkspace object
private ORiN2.ManagedCAO.CCaoWorkspace m_caoWorkspace = null;
// Variable for CaoController object
private ORiN2.ManagedCAO.CCaoController m_caoController = null;
// Variable for CaoVariable object
private ORiN2.ManagedCAO.CCaoVariable m_varLastResult = null;
```

(2) Create a CaoEngine object.

Create a CaoEngine object by using "new" keyword.

Example of use (C#)

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

(3) Get or create a CaoWorkspace object.

When a CaoEngine object is created, one CaoWorkspaces object and one CaoWorkspace object are created by default. Below is an example code that creates a new CaoWorkspace object and the default CaoWorkspace.

Example of use (C#)

```
// Create a 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 be used and the parameters required to use it. The ΩEye provider specifies the URL of the ΩEye server to specify the ΩEye to connect to. An example code is shown below.

Example of use (C#)

```
// Create a CaoController object
this.m_caoController = this.m_caoWorkspace.AddController("OmegaEye",
                                                         "CaoProv.SOFIX.SOFIXCAN.OmegaEye.V3",
                                                         "",
                                                         "Uri = http://192.168.0.1/OMEGA-EYE-API");
```

(5) Create a CaoVariable object.

To create a CaoVariable object, set the provider name to be used and the parameters required to use it. To get the latest recognition result, the variable name is set to "LAST_RESULT1" and the parameter is unspecified to use the default value. An example code is shon below.

Example of use (C#)

```
// Create a CaoVariable object
this.m_varLastResult = this.m_caoController.AddVariable("LAST_RESULT1","");
```

4.1.1.2. Get the latest recognition results

Get the latest recognition results from ΩEye. The ΩEye provider makes a connection with ΩEye when reading a variable and obtains the value. An example code is shown below.

Example of use (C#)

```
// Read variable
string[] values = this.m_varLastResult.Value as string[];
for (var i = 0; i < values.Length; i++)
{
    // Get comma-separated data
```

```
    var splitData = values[i];
}
```

4.1.1.3. Delete objects

Post-processing involves deleting the created object and deleting the object to be deleted from the collection class that manages the object. Note that if you use ORiN.ManagedCAO, it is not necessary to delete it explicitly. An example code is shown below.

Example of use (C#)

```
// Delete all objects in CaoEngine
this.m_caoEngine.Dispose();
// Delete CaoEngine
this.m_caoEngine = null;
```

5. ΩEye provider error code

This provider has the following unique error codes masked by 0x8011****. (Refer to Table 5-1 Original error code table.)

For ORiN2 common errors, please refer to the Error Codes chapter of the ORiN2 Programming Guide.

Table 5-1 Original error code table

Error number	Description
0x80110001	Required options are not specified.
0x80110002	An option was specified that cannot be parsed.
0x80110003	An option outside the value range was specified.
0x80111001	An abnormal response from ΩEye has occurred.
0x80111002	A timeout error has occurred.
0x80110301	An unsupported variable was specified.
0x80110302	Set operation was performed on a variable that does not support Set operation.

In addition, this provider masks the HTTP responder code with "0x8010****" and returns it.

Appendix. A. Request URI Correspondence Table

Variable name	get_Value	put_Value
@STATUS	get_state_orin	---
@SYSTEM_INFO	get_system_info_orin	---
LAST_RESULT	get_latest_result_orin	---
RESULTS	get_results_orin	---

Appendix. B. About Profiles

A profile is a file that contains configuration information used for recognition. Profiles are a feature of the Plus and Windows version only. The Plus and Windows version retains up to 10 profiles, and "camera selection" and "checkpoint setting" are performed in each profile.