Murata Manufacturing
Wireless Sensing Solution Gateway Provider
User's Guide

Version 1.0.0

February 8, 2023

NOTE:

[Revision History]

Version	Date	Content
1.0.0	2022-03-28	First edition
	2022-10-04	Provider name and supported model corrected
	2023-02-08	Errors have been corrected.

[Compatible models]

Model	Version	Notes
Gateway for wireless sensing solutions		

[Operation check model]

Model	Version	Notes
Gateway for wireless sensing solutions		The sensors whose connections were checked are as follows LBAC0ZZ1MT-247<466> LBAC0ZZ1LZ-123<448> LBAC0ZZ1AN-783<578> LBAC0ZZ1MU-270<464> LBAC0ZZ1TF-474<556>

Contents

1. Introduction

This document is a user's guide for the provider who connects to the wireless sensing solution gateway of Murata Manufacturing Co., Ltd. and receives measurement information from the wireless sensor unit via the wireless sensing solution gateway. Fig. 1-1 shows the overall configuration of this provider and the device. From here on, the provider is called the wireless sensing solution gateway provider.

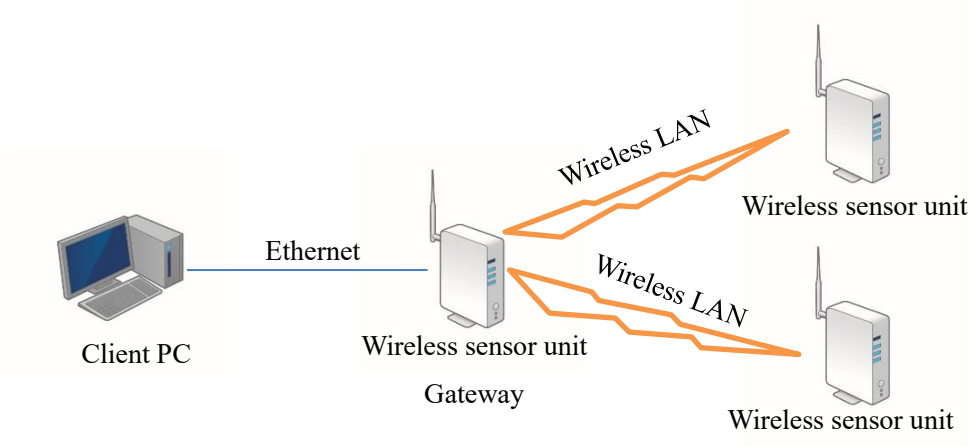

Fig. 1-1 Configuration Diagram

Fig. 1-2 shows the correspondence between this provider and each device.
(* This is an example. It does not represent everything.)

Fig. 1-2 Provider configuration and device information

2. Setting Up Your Environment for Application Development

2.1. Connection between the wireless sensing solution gateway and the client PC

The wireless sensing solution gateway and the client PC communicate via UDP.

When connecting, connect the wireless sensing solution gateway directly to the client PC, or connect the network hub by relaying the network hub.

When communicating beyond the router, it is necessary to set NAT so that the message from the wireless sensing solution gateway can be received correctly.

2.2. Setting up a PC development environment

2.2.1. Installing the Wireless Sensing Solution Gateway Provider Manually

If you install the wireless sensing solution gateway provider manually, you must register the registry as follows: To register the registry, start the command prompt with administrator privileges and execute regsvr32 command. When executing the command, either move to the path where the file is located or specify the file path.

Table 2-1 Wireless Sensing Solution Gateway Providers

File name	CaoProvMurataWirelessGateway.dll
ProgID	CaoProv.Murata.WirelessGateway
Registry registration	Regsvr32 CaoProvMurataWirelessGateway.dll
Deletion of Registry Registration	Regsvr32 /u CaoProvMurataWirelessGateway.dll

3. Command Reference

3.1. Method/Property List

Table 3-1: List of methods and properties

Category	Methods/Properties[1]		Function	Reference
CaoWorkspace				
	AddController	M	Connected to controller	P.9
CaoController				
	Index	P	Obtaining the Controller Number	P.10
	Name	P	Get Controller Name	P.10
	GetVariableNames	M	Get a list of variable names that can be connected	P.11
	Variables	P	Retrieving Variable Collections Holded by the Controller	P.11
	AddVariable	M	Adding Variable Objects	P.11
	OnMessage	E	Message reception event	P.11
CaoVariable				
	Index	P	Obtaining Variable Numbers	P.12
	Name	P	Retrieving Variable Names	P.12
	Value	P	Get/set value	P.12

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

3.2. Method properties

3.2.1. CaoWorkspace classes

3.2.1.1. AddController method

Add controller objects to CaoWorkspace. The wireless sensing solution gateway provider refers to the parameters passed when AddController method is executed and connects to the corresponding wireless sensing solution gateway. The following are the specifics of AddController method:

Format

```
AddController
(
"<controller name>",                    // Controller name (optional)
"CaoProv.Murata.WirelessGateway",       // Provider name (fixed)
"<machine name>",                       // Provider execution machine name (unused)
"<Option>"                              // Option string (optional)
)
```

Option

The following options are specified in the option string: The option string is a string consisting of the following options separated by a comma (,).

Option	Required	Description	Value Range	Default value
CONN	✓	Specify the IP address and port number of the wireless sensing solution gateway.	-	-

Sample usage (C#)

```
// Engine
ORiN2.ManagedCAO.CCaoEngine engine = new ORiN2.ManagedCAO.CCaoEngine();
// Workspace
ORiN2.ManagedCAO.CCaoWorkspace workspace = engine.AddWorkspace("NewWrks", "");
// Controller
ORiN2.ManagedCAO.CCaoController controller= workspace.AddController(
    "Gateway", "CaoProv.Murata.WirelessGateway", "",
    "CONN=UDP:192.168.0.1:55061");
```

3.2.1.1.1. CONN Optional

Use CONN option to specify the IP address and port number of the wireless sensing solution gateway to which you want to connect.

Each element is separated by a ":".

The meaning of each element is as follows.

Since the measurement start command is sent to the wireless sensing solution gateway when the provider is started, measurement information can be received even if the IP address of the own station and the port number of the own station are omitted. However, if the wireless sensing solution gateway is restarted, measurement information is sent to the destination IP and destination port set in WsnManager, so it is recommended to set the IP address of the own station and the port number of the own station according to the destination IP and destination port set in WsnManager.

Item number	Meaning	Required	Default value
0	Protocol. "UPD" fixed	✓	-
1	Destination iP address	✓	-
2	Number of the specific port	-	55061
3	Own station's IP address	-	Optional
4	Own port number	-	Automatic

Example) CONN=UDP:192.168.1.100:55061

3.2.2. CaoController classes

3.2.2.1. Index Properties

Gets the controller number as a Long type (4-byte integer type). This number is used by the corresponding controller to identify the controller in the controller collection held by CaoWorkspace.

Sample usage (C#)

```
// Get Index
long index = controller.Index;
```

3.2.2.2. Name Properties

Retrieves the controller name specified by AddController method of CaoWorkspace class.

Sample usage (C#)

```
System.Diagnostics.Debug.WriteLine(controller.Name);
```

3.2.2.3. GetVariableNames method

Gets a list of variable names that can be connected.

Format

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

Option

The wireless sensing solution gateway provider does not use options.

Sample usage (C#)

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

3.2.2.4. Variables Properties

Gets a collection of variables that the controller holds.

Sample usage (C#)

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

3.2.2.5. AddVariable method

Adds a variable object to CaoController. Only the variable names shown in 3.3.2 can be used.3.3.2 AddVariable is specifi ed as follows.

Format

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

3.2.2.6. OnMessage event

The wireless sensing solution gateway can receive the measured data received from the wireless sensor unit as a OnMessage event. See 3.4 for the events that can be received.

3.2.3. CaoVariable classes

3.2.3.1. Index Properties

Gets the variable number as a Long type (4-byte integer type). This number is used to identify the variable in the variable collection held by CaoController.

Sample usage (C#)

```
// Get Index
Int index = caoVar.Index;
```

3.2.3.2. Name Properties

Retrieves the variable name specified by AddVariable method of CaoContrller class.

Sample usage (C#)

```
System.Diagnostics.Debug.WriteLine(caoVar.Name);
```

3.2.3.3. Value Properties

Get information about the provider. For details, refer to section 3.3.

3.3. Variable list

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

3.3.1. System and User Variables

Only system variables are present in the Wireless Sensing Solution Gateway Provider.

System Variables

A variable that accesses only the information in the object that holds the variable. In the Wireless Sensing Solution Gateway Provider, the values that can be retrieved in the System Variables are static data that provides information about the provider. System variables are preceded by "@".

e.g. provider version, device manufacturer

3.3.2. CaoController class-variable

Variable Name	Description	Value		Reference
		Get	Put	
@MAKER_NAME	Obtain the manufacturer's name.	✓	-	P.12
@VERSION	Get the DLL version.	✓	-	P.13

3.3.2.1. @MAKER_NAME

Obtain the manufacturer's name.

Data type

Type Description	
VT_BSTR	Obtain the manufacturer's name.

Sample usage (C#)

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

3.3.2.2. @VERSION

Gets the DLL version.

Data type

Type Description	
VT_BSTR	Get the DLL version.

Sample usage (C#)

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

3.4. Event list

The wireless sensing solution gateway can receive the measured data received from the wireless sensor unit as a OnMessage event.

The measured data from the wireless sensor unit is received as a message with a Number of 0.

3.4.1.1. Measurement Information Message

This message notifies the measurement information received by the wireless sensing solution gateway from the wireless sensor unit.

Value Property Data Types

Type Description		
VT_ARRAY | VT_VARIANT		
0	VT_UI2	ID of the wireless sensor unit that sent the measurement information.
1	VT_I2	The signal strength (RSSI) when the measured data is received.
2	VT_UI1	Sensor type. Refer to 3.4.1.1.1 for details.

Type	Description	
3	VT_VARIANT | VT_ARRAY	Measured value. The order of internal data and the type of value differ for each sensor type. Elements without measurements will contain VT_EMPTY. For details, refer to 3.4.1.1.2 to 3.4.1.1.11.
4	VT_UI1 | VT_ARRAY	A measurement unit. The number of elements is the same as the measurement value, and the unit of the measurement value is the same unit of the element number as the measurement value. Refer to xxx for the meaning of the unit.

Sample usage (C#)

```
///<summary>
/// Example of Message Reception Processing Method
///</summary>
///<param name="sender>>Event source</param>
///<param name="e">Eventargument</param>
private void OnMessage(object sender, OnMessageEventArgs e)
{
    object[] dataArray = e.Message.Value as object[];
    if (dataArray != null)
    {
        ushort unitId = Convert.ToUInt16(dataArray[0]);
    }
}
```

3.4.1.1.1. List of sensor types

The sensor type is as follows.

Sensor type	Sensor type name	Last three characters of the product name[2]
0x01	Temperature and humidity	1AN
0x07	3 Temperature/Thermocouple	1EM/1PF
0x10	Current and pulse	1MU
0x13	Voltage and pulse	1RU
0x12	CT	1NT/1MT
0x09	Vibration	1LZ
0x18	Vibration (rate)	1TF

[2] Xxx part of the product-name LBAC0ZZxxx-***

0x1B	3 current	1ZU
0x1C	3 Voltage	1ZV
0x1D	3 contacts	1ZS

3.4.1.1.2. Measured value of temperature and humidity (0x01)

Measured values of temperature and humidity are as follows.

Element number	Value Type	Meaning of the values
0	VT_R8	Power supply voltage
1	VT_R8	Temperature
2	VT_R8	Humidity

3.4.1.1.3. 3 Measured value of temperature/thermocouple (0x07)

The temperature/thermocouple measurements are as follows:

Element number	Value Type	Meaning of the values
0	VT_R8	Power supply voltage
1	VT_R8	Temperature 1
2	VT_R8	Temperature 2
3	VT_R8	Temperature 3

3.4.1.1.4. Measured value of current and pulse (0x10)

Measured values of current and pulse are as follows.

Element number	Value Type	Meaning of the values
0	VT_R8	Power supply voltage
1	VT_R8	Counter
2	VT_R8	Current
3	VT_R8	Pulse count
4	VT_R8	Previous current
5	VT_R8	Previous pulse count

3.4.1.1.5. Measured value of voltage pulse (0x13)

The measured values of voltage and pulse are as follows.

Element	Value Type	Meaning of the values

number		
0	VT_R8	Power supply voltage
1	VT_R8	Counter
2	VT_R8	Voltage
3	VT_R8	Pulse count
4	VT_R8	Previous voltage
5	VT_R8	Previous pulse count

3.4.1.1.6. CT (0x12) measurements

The CT measurements are as follows:

Element number	Value Type	Meaning of the values
0	VT_R8	Power supply voltage
1	VT_R8	Counter
2	VT_R8	Current 1
3	VT_R8	Current 2
4	VT_R8	Energy harvesting
5	VT_R8	Last current 1
6	VT_R8	Last current 2
7	VT_R8	Last energy harvesting

3.4.1.1.7. Measured value of vibration (0x09)

Measured values of vibration are as follows.

Element number	Value Type	Meaning of the values
0	VT_R8	Power supply voltage
1	VT_R8	Peak Frequency 1
2	VT_R8	Peak acceleration 1
3	VT_R8	Peak Frequency 2
4	VT_R8	Peak acceleration 2
5	VT_R8	Peak Frequency 3
6	VT_R8	Peak acceleration 3
7	VT_R8	Peak Frequency 4
8	VT_R8	Peak acceleration 4
9	VT_R8	Peak Frequency 5
10	VT_R8	Peak acceleration 5

11	VT_R8	Acceleration RMS
12	VT_R8	Kurtness
13	VT_R8	Temperature

3.4.1.1.8. Measured value of vibration (speed) (0x18)

Measured values of vibration (rate) are as follows.

Element number	Value Type	Meaning of the values
0	VT_R8	Power supply voltage
1	VT_R8	Peak Frequency 1
2	VT_R8	Peak acceleration 1
3	VT_R8	Peak Frequency 2
4	VT_R8	Peak acceleration 2
5	VT_R8	Acceleration RMS
6	VT_R8	Speed Peak Frequency 1
7	VT_R8	Peak Speed 1
8	VT_R8	Speed Peak Frequency 2
9	VT_R8	Peak Speed 2
10	VT_R8	Speed Peak Frequency 3
11	VT_R8	Peak Speed 3
12	VT_R8	Speed RMS
13	VT_R8	Kurtness
14	VT_R8	Temperature

3.4.1.1.9. 3 Measured current (0x1B)

3 Measured values of current are as follows.

Element number	Value Type	Meaning of the values
0	VT_R8	Power supply voltage
1	VT_BSTR	Serial number
2	VT_R8	Current value 1
3	VT_R8	Current value 2
4	VT_R8	Current value 3

3.4.1.1.10. 3 Measured voltage (0x1C)

3 Measured values of voltage are as follows.

Element number	Value Type	Meaning of the values
0	VT_R8	Power supply voltage
1	VT_BSTR	Serial number
2	VT_R8	Voltage value 1
3	VT_R8	Voltage value 2
4	VT_R8	Voltage value 3

3.4.1.1.11. Measured value of 3 contacts (0x1D)

3 Measured values of voltage are as follows.

Element number	Value Type	Meaning of the values
0	VT_R8	Power supply voltage
1	VT_BSTR	Serial number
2	VT_R8	Detection edge 1
3	VT_R8	Edge count 1
4	VT_R8	Detection edge 2
5	VT_R8	Edge count 2
6	VT_R8	Detection edge 3
7	VT_R8	Edge count 3

3.4.1.1.12. Unit list

The following is a list of units.

Unit code	Meaning	Unit
0x01	Duty ratio	%
0x02	Pressure	Pa
0x03	Position_latitude	
0x04	Position_longitude	
0x05	Inductance	H
0x06	Energy	J
0x07	Count	Time
0x08	Angular acceleration	Rad/s^2
0x09	Diffusivity	m^2/s
0x0A	Angular velocity	Rad/s
0x0B	Angle	°
0x0C	Acceleration	m/s^2
0x0D	Brightness	Cd/m^2
0x0E	Absorbed Dose	Gy
0x0F	Absorbed dose rate	Gy/s
0x10	Refractive index	-
0x11	Enzymatic activity	Kat
0x12	Enzymatic activity level	Kat/m^3
0x13	Light bundle	Lm
0x14	Light intensity	Cd
0x15	Conductance	S
0x16	Size	Byte
0x17	Magnetic field	A/m
0x18	Time	Sec
0x19	Time	Hour
0x1A	Time	Minute
0x1B	Time	Second

Unit code	Meaning	Unit
0x1C	Time	Millsecond
0x1D	Work Rate	W
0x1E	Magnetic flux	Wb
0x1F	Magnetic flux density	T
0x20	Weight	Kg
0x21	Mass absorption coefficient	m^2/kg
0x22	Mass density	Kg/m^3
0x23	Molality	Mol/kg
0x24	Mass flow rate	Kg/s
0x25	Mass flow density	$Kg/m^2 \cdot s$
0x26	Frequency	Hz
0x27	Dose	C/kg
0x28	Illuminance	Lx
0x29	Numeric value	-
0x2A	Celsius temperature	℃
0x2B	Dose equivalent	Sv
0x2C	Relative humidity	%RH
0x2D	Volume	m^3
0x2E	Volume energy	J/m^3
0x2F	Volumetric flow rate	m^3/s
0x30	Power	N
0x31	Moment of force	N・m
0x32	Potential difference (voltage)	V
0x33	Electric charge	C
0x34	Electric field	V/m

Unit code	Meaning	Unit
	strength	
0x35	Charge density	C/m^3
0x36	Electrical resistance	Ohm
0x37	Electric capacity	F
0x38	Electric flux density	C/m^2
0x39	Current	A
0x3A	Current density	A/m^2
0x3B	Electric energy	kWh
0x3C	Permeability	H/m
0x3D	Conductivity	S/m
0x3E	Kinematic viscosity	m^2/s
0x3F	Length	M
0x40	Sunlight volume	kW/m^2
0x41	Thermal conductivity	W/ (m・K)
0x42	Heat capacity	J/K
0x43	Thermodynamic temperature	K
0x44	Areic heat flow rate	W/m^2
0x45	Viscosity	Pa・s
0x46	Density	Ppm
0x47	Wave number	m^{-1}
0x48	Speed	m/s
0x49	Specific energy	J/kg
0x4A	Relative permeability	-
0x4B	Specific heat capacity	J/ (kg・K)
0x4C	Specific volume	m^3/kg
0x4D	Surface pressure	N/m

Unit code	Meaning	Unit
0x4E	Surface charge	C/m^2
0x4F	Material content	Mol
0x50	Plane angle	Rad
0x51	Radiance	W/ (m^2・sr)
0x52	Radiation intensity	W/sr
0x53	Radioactivity	Bq
0x54	Density	Kg/m^3
0x55	Area	m^2
0x56	Areal density	Kg/m^2
0x57	Morphological energy	J/mol
0x58	Morph entroby	J/ (mol・K)
0x59	Molar Volume	m^3/mol
0x5A	Molar conductivity	S・m^2/mol
0x5B	Dielectric constant	F/m
0x5C	Force product	N・s
0x5D	Solid angle	Sr
0x5E	Volume density	Mol/m^3
0x5F	Radiation Dose	uSv
0x60	Body sensor	-
0x61	Magnetic sensor	-
0x62	Current	mA
0x63	Root-mean-square value of acceleration	m/s^2
0x64	Atmospheric pressure	hPa
0x65	ADC output	-
0x66	Kurtality	-
0x68	No unit	-
0x69	Electric power	W
0x6A	Flow rate	L/min

Unit code	Meaning	Unit
0x6B	Volume	L
0x71	Angle	Rad
0x72	Speed	Mm/s
0x73	Displacement	Mm

Unit code	Meaning	Unit
0x74	Velocity RMS	Mm/s
0xFA	Serial No.	-
0xFE	Range over	-
0xFF	Error	-

4. Programming by wireless sensing solution gateway provider

The Wireless Sensing Solution Gateway Provider allows you to connect the Client PC to the Wireless Sensing Solution Gateway as follows:

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

After connecting to the wireless sensor unit, it is possible to acquire the measurement information sent from the wireless sensor unit at OnMessage event of CaoController.

4.1. Sample Programming to Display Temperature and Humidity Meter Measurement Information

Here is an example program of C# using ManagedCAO that displays the measured data of the temperature and hyphenation meter as an example. Table 4-1 describes the requirements of the sample program, and Fig. 4-1 describes the flow of the sample program.

Table 4-1 Sample program requirements

Requirements	Description
Host	Connect via UDP
	Read destination IP address and port from console
Process Description	If the measured information of the temperature/hyphenation meter was received at the time of OnMessage reception, the content of the information is displayed on the window.

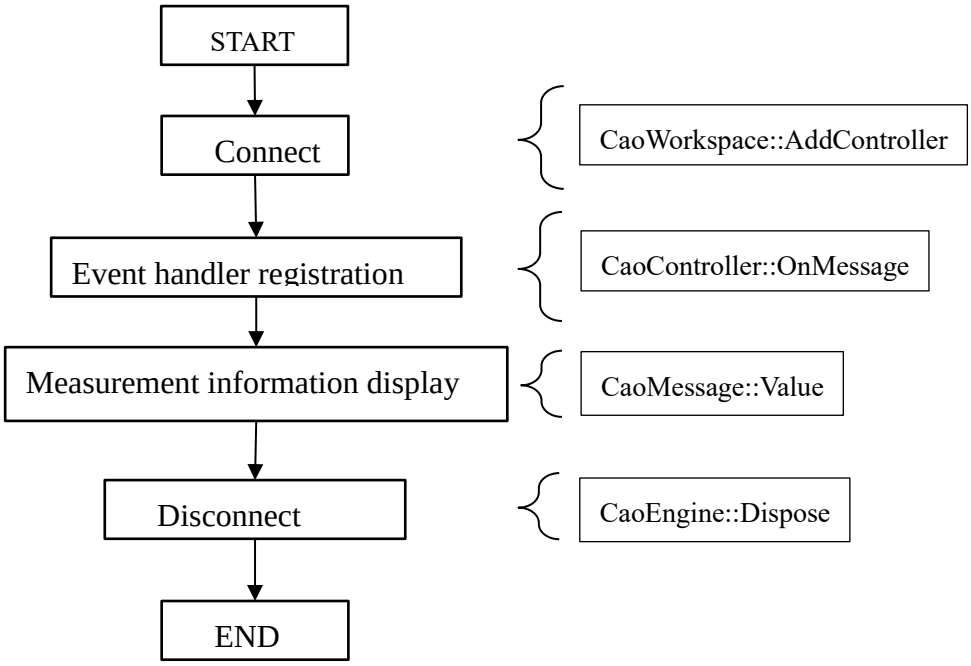

Fig. 4-1 Sample program flow

Specific codes are given in the following sections.

4.1.1. Sample program

The following is an overview of the sample program.

Sample **Program.cs**

```
using System;
using ORiN2.ManagedCAO;

namespace MurataWirelessGatewaySample
{
    ///<summary>
    /// Program that displays the measurement information of the temperature and hyphenation meter on the screen
    ///</summary>
    class Program
    {
        ///<summary>
        /// Main processing
        ///</summary>
        ///<param name="args">runtime parameter</param>
        static void Main(string[] args)
```

```
{
    try
    {
        // Causes CONN = UDP: to be followed by an optional entry
        string option = "CONN=UDP:";
        Console.Write(option);
        option += Console.ReadLine();

        // Create a CaoEngine
        // Automatically when exiting the scope by enclosing it in using
        // Dispose is executed and released
        using (CCaoEngine engine = new CCaoEngine())
        {
            // Create a CaoController
            // engine.Workspaces[0] is generated automatically
            CCaoController controller = engine.Workspaces[0].AddController(
                "Gateway", "CaoProv.Murata.WirelessGateway", null, option);
            // Registering OnMessage event handlers
            controller.OnMessage += Controller_OnMessage;

            Console.WriteLine("Hit any key to exit.");
            Console.ReadKey();

            // Release OnMessage event handler
            controller.OnMessage -= Controller_OnMessage;
            // Dispose is called to exit scope at} in the following line
        }
    }
    catch(Exception ex)
    {
        // Output exception content to console
        Console.WriteLine(ex);
    }
}

///<summary>
/// Message Handling Functions
```

```
///</summary>
///<param name="sender">Source object</param>
///<param name="e">Eventargument</param>
private static void Controller_OnMessage(object sender, OnMessageEventArgs e)
{
    try
    {
        object[] values = e.Message.Value as object[];
        // Do not process non-purpose messages
        if (e.Message.Number != 0
            || values == null
            || values.Length != 5
            || Convert.ToByte(values[2]) != 0x01)
        {
            return;
        }

        ushort sensorId = Convert.ToUInt16(values[0]);
        short rssi = Convert.ToInt16(values[1]);
        byte sensorTypeId = Convert.ToByte(values[2]);
        object[] sensorValues = values[3] as object[];
        if (sensorValues == null || sensorValues.Length != 3)
        {
            Console.WriteLine("invalid message.");
            return;
        }
        double supplyVoltage = Convert.ToDouble(sensorValues[0]);
        double temperature = Convert.ToDouble(sensorValues[1]);
        double humidity = Convert.ToDouble(sensorValues[2]);

        Console.WriteLine(
            "sensorId:{0:X4}, rssi:{1}, sensorTypeId:{2:X2}, " +
            supplyVoltage:{3}, temperature:{4}, humidity:{5}",
            sensorId, rssi, sensorTypeId, supplyVoltage, temperature, humidity);
    }
    catch (Exception ex)
    {
```

```
                    // Output exception content to console
                    Console.WriteLine(ex);
                }
            }
        }
    }
```

4.1.1.1. Connection

To connect to the wireless sensing solution gateway, take the following steps.

(1) Creates a CaoEngine object and a CaoController object.

When you are finished using CaoEngine object, you must call CaoEngine.Dispose() to free the object. Using clause automatically invokes Dispose when you leave the scope to prevent you from forgetting to release it.

A CaoWorkspace is automatically generated in CaoEngine.Workspaces[0] when CaoEngine is generated. Calling AddController of this CaoWorkspace creates a CaoController object and connects it to the wireless sensing solution gateway.

Sample usage (C#)

```
// Create a CaoEngine
// Automatically when exiting the scope by enclosing it in using
// Dispose is executed and released
using (CCaoEngine engine = new CCaoEngine())
{
    // Create a CaoController
    // engine.Workspaces[0] is generated automatically
    CCaoController controller = engine.Workspaces[0].AddController(
        "Gateway", "CaoProv.Murata.WirelessGateway", null, option);
```

(2) Registers event handlers to receive OnMessage events in CaoController object.

Registers event handlers to receive events in OnMessage event of CaoController object.

The type of event handler to register is as follows.

```
public delegate void OnMesssageEventHandler(object sender, OnMessageEventArgs e);
```

Sample usage (C#)

```
controller.OnMessage += Controller_OnMessage;
```

4.1.1.2. Event processing

(1) Describes the action to be taken when a OnMessage event is received.

When the measured data is received from the wireless sensor unit, OnMessage of CaoController occurs.

Since the received measurement data is stored in Message property of the event parameter, process the measurement data.

In the sample program, only the measured data from the temperature and hygrometer is output to the console.

Sample usage (C#)

```
///<summary>
/// Message Handling Functions
///</summary>
///<param name="sender">Source object</param>
///<param name="e">Eventargument</param>
private static void Controller_OnMessage(object sender, OnMessageEventArgs e)
{
    try
    {
        object[] values = e.Message.Value as object[];
        // Do not process non-purpose messages
        if (e.Message.Number != 0
            || values == null
            || values.Length != 5
            || Convert.ToByte(values[2]) != 0x01)
        {
            return;
        }

        ushort sensorId = Convert.ToUInt16(values[0]);
        short rssi = Convert.ToInt16(values[1]);
        byte sensorTypeId = Convert.ToByte(values[2]);
        object[] sensorValues = values[3] as object[];
        if (sensorValues == null || sensorValues.Length != 3)
        {
```

```
                    Console.WriteLine("invalid message.");
                    return;
                }
                double supplyVoltage = Convert.ToDouble(sensorValues[0]);
                double temperature = Convert.ToDouble(sensorValues[1]);
                double humidity = Convert.ToDouble(sensorValues[2]);
                Console.WriteLine(
                    "sensorId:{0:X4}, rssi:{1}, sensorTypeId:{2:X2}, " +
                    supplyVoltage:{3}, temperature:{4}, humidity:{5}",
                    sensorId, rssi, sensorTypeId, supplyVoltage, temperature, humidity);
            }
            catch (Exception ex)
            {
                // Output exception content to console
                Console.WriteLine(ex);
            }
        }
```

4.1.1.3. Disconnect

To disconnect from the controller, you can cancel the registered event handler and delete the object to be deleted from the collection class that manages the object. However, you do not need to explicitly describe the deletion process because when you use ManagedCAO, Dispose invocation of CaoEngine automatically deletes all objects.

Sample usage (C#)

```
                // Release OnMessage event handler
                controller.OnMessage -= Controller_OnMessage;
                // Dispose is called to exit scope at} in the following line
            }
```