Parker
WirelessMSCL providers
User's Guide

Version 1.0.1

July 14, 2022

NOTE:

[Revision History]

Version	Date	Content
1.0.0	2021-08-27	First edition
1.0.1	2022-07-14	Changed LORD company name in document to Parker Changed the acquisition value of @MAKER_NAME variable to Parker

[Compatible models]

Model	Version	Notes
WSDA-2000	---	USB connection only
WSDA-Base-200 (USB)	---	---
WSDA-Base-101 (Analog)	---	---
WSDA-Base-104 (USB)	---	---

[Operation check model]

Model	Version	Notes
WSDA-Base-200 (USB)	6.44079	

Contents

1. Introduction

This user's guide is for providers who collect data from Parker's wireless nodes (hereinafter referred to as nodes) to wireless gateways that collect data such as channels (hereinafter referred to as gateways). Fig. 1-1 shows the overall configuration of this provider and the device. This provider uses Parker's MicroStrain communication library (MicroStrain Communication Library(MSCL) to connect to the gateway. MSCL uses serial communication (USB connection) to communicate with the gateway. The providers are referred to as WirelessMSCL providers.

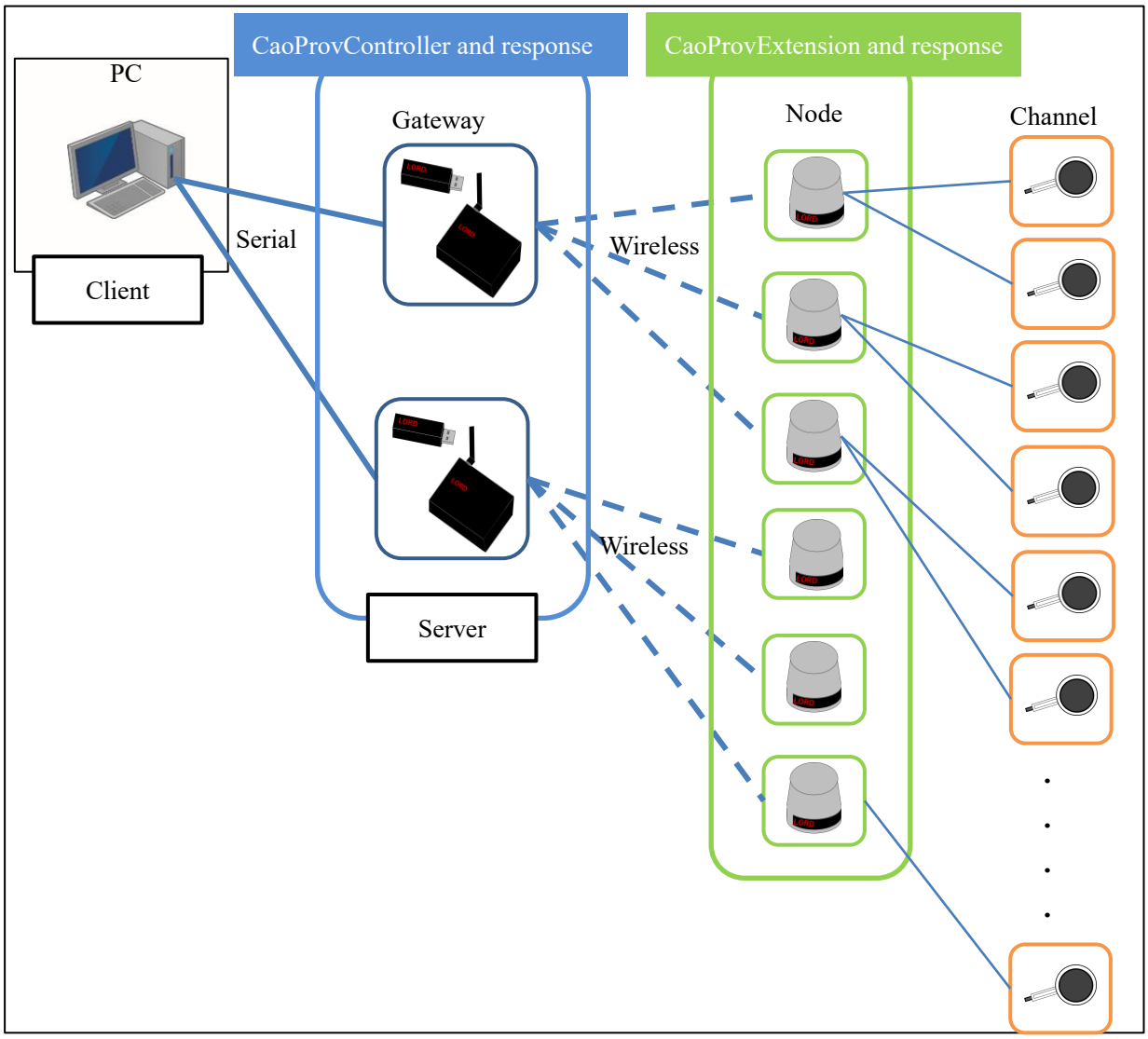


Fig. 1-1 Configuration Diagram

Fig. 1-2 shows the correspondence between this provider and each device.

(* This is an example. It does not represent everything.)

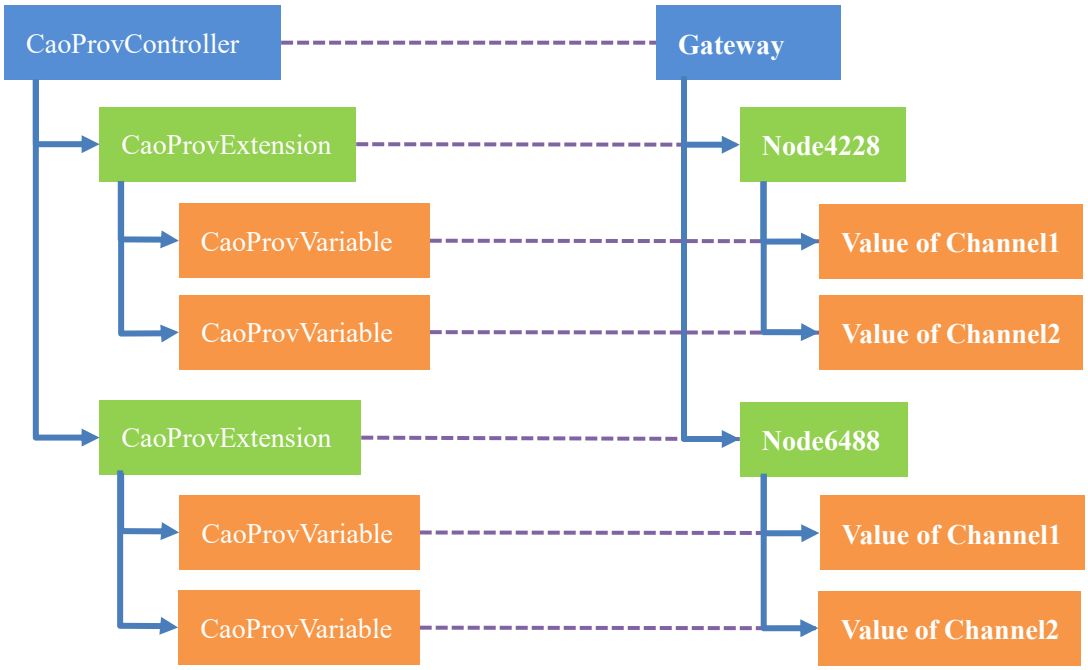


Fig. 1-2 Provider configuration and device information

1.1. About Your Library

WirelessMSCL providers use MSCL, an OSS from Parker, to communicate with wireless gateways.

For more MSCL, please visit:

[Related site link]

https://www.microstrain.com/software/mscl

[MSCL Rights and Licensing]

https://github.com/LORD-MicroStrain/MSCL/blob/master/LICENSE

2. Setting Up Your Environment for Application Development

2.1. Preparing the Gateway to Connect to the Client PC

2.1.1. Connecting to the Client PC Gateway

The gateway and the client PC connect via serial communication. A communication driver is required for connection. For details on installing the communication driver, refer to the following website.

https://github.com/LORD-MicroStrain/Drivers

2.1.2. Installing SensorConnect Application

Parker's configuration application "SensorConnect" is required when configuring gateways and nodes. Download SensorConnect from Parker website at the URL below and install it. The communication driver is installed at the same time that the application is installed.

https://www.microstrain.com/software

2.1.3. Gateway and Node Communication Protocol and Frequency Band Settings

Use Parker's "SensorConnect" configuration application to set the communication protocol and frequency band of the gateway and node to be connected according to the environment. The figure below shows a conceptual diagram of the frequency band.

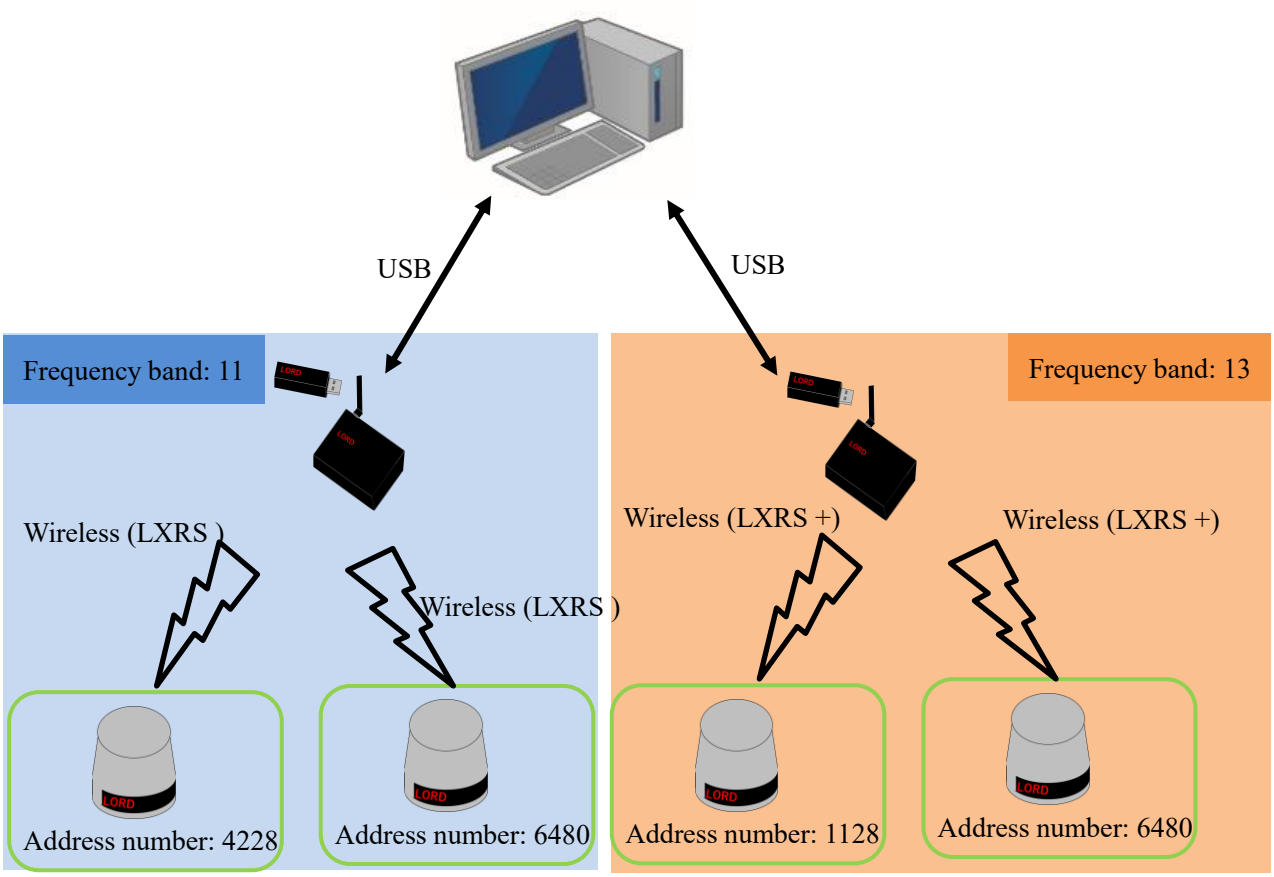


Fig. 2-1 Relationship between communication protocol and frequency band

Communications protocols

A method of wireless communication between a gateway and a node. There are LXRS and + LXRS communication methods.

- **LXRS**

LXRS is designed to override network throughput and operate at communication distances of up to 2km and network throughput of up to 4ksamples per second.

- **LXRS+**

LXRS+ is designed to override network throughput and operate at network throughput of up to 16ksamples/sec at a distance of up to 400m.

Frequency band

This is the frequency band assigned to the radio communication between the gateway and the node. Gateways

and nodes in the same frequency band can communicate.

Frequency band	Frequency (Bandwidth)
11	2.405 Ghz
12	2.410 Ghz
13	2.415 Ghz
14	2.420 Ghz
15	2.425 Ghz
16	2.430 Ghz
17	2.435 Ghz
18	2.440 Ghz
19	2.445 Ghz
20	2.450 Ghz
21	2.455 Ghz
22	2.460 Ghz
23	2.465 Ghz
24	2.470 Ghz
25	2.475 Ghz
26	2.480 Ghz

2.2. Setting up a PC development environment

2.2.1. Preparation MSCL

WirelessMSCL providers use MSCL provided by Parker Corporation. MSCL is installed automatically when ORiN2 is installed.

Table2-1 MSCL to be used

DLL
MSCL.dll
MSCL_Managed.dll

2.2.2. Installing WirelessMSCL Providers Manually

If you install WirelessMSCL providers manually, you must register the registry as shown below. The file format is EXE and is dynamically loaded when used by the CAO engine. To use WirelessMSCL providers, you must register as shown in Table2-2 WirelessMSCL Providers. RegistAsm.bat and UnregistAsm.bat are located in DotNet¥BAT folder under the folder where you installed ORiN2SDK.

Table2-2 WirelessMSCL Providers

File name	CaoProvLORDWirelessMSCL.exe
ProgID	CaoProv.LORD.WirelessMSCL
Registry registration	RegistAsm.bat CaoProvLORDWirelessMSCL.exe
Deletion of Registry Registration	UnregistAsm.bat CaoProvLORDWirelessMSCL.exe

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.13
CaoController				
	GetVariableNames	M	Get a list of variable names that can be connected	P.16
	Variables	P	Retrieving Variable Collections Holded by the Controller	P.16
	AddExtension	M	Adding an Expansion Board Object	P.16
	AddVariable	M	Adding Variable Objects	P.19
	Execute	M	Execute Extended Commands	P.20
	OnMessage	E	Message reception event	P.20
CaoExtenstion				
	GetVariableNames	M	Get a list of variable names that can be connected	P.20
	Variables	P	Retrieving Variable Collections Retained by a Task	P.21
	AddVariable	M	Adding Variable Objects	P.21
	Execute	M	Execute Extended Commands	P.21
CaoVariable				
	Value	P	Get/set value	P.21

3.2. Method properties

3.2.1. CaoWorkspace classes

3.2.1.1. **AddController method**

In CaoWorkspace, add a controller object. WirelessMSCL providers refer to the parameters passed when AddController method is executed and connect to the appropriate gateway. The following are the specifics of AddController method:

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

Format

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

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 connection destination information.	-----	-----
Timeout	-	Specifies the timeout in ms.	1 - 65535	300
DataCollectionCycle	-	Specifies the interval (in ms) at which data is collected from the gateway. Refer to 3.2.1.1.2 for details.	1 - 30000	10
IsBroadcastSetToIdle	-	Execute the command to make the node idle by broadcast at the time of connection. Refer to 3.2.1.1.3 for details.	True,false	True

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(Basestation1,
                                        "CaoProv.LORD.WirelessMSCL",
                                        "",
                                        "Conn = COM:1,Timeout = 200, DataCollectionCycle = 20");
```

3.2.1.1.1. **Conn Optional**

The following is a Conn optional connection parameter string:

・**Serial communication**

"Conn=COM:<COM Port>"

<COM Port> : COM port number. '1' - COM1, '2' - COM2, ...

3.2.1.1.2. **Periodically gather data from the gateway**

WirelessMSCL providers regularly collect data from the gateway after AddController, as shown in Fig. 3-1. When synchronous sampling is started in 3.3.1.2, the data acquired by the node in synchronous sampling is transmitted to the gateway. The gateway retains the transmitted data once. This provider periodically collects data from the gateway and periodically samples the data of the retained gateway at DataCollectionCycle optionspecified at the time of AddController.

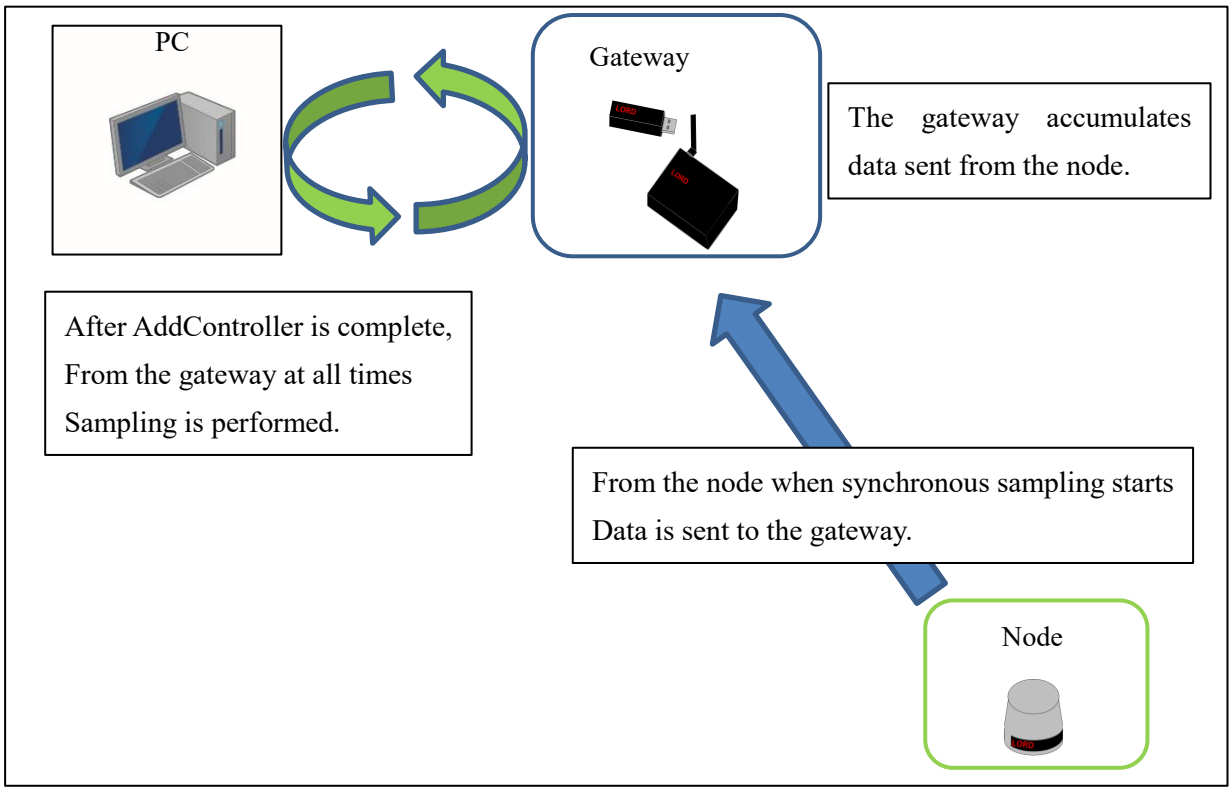


Fig. 3-1 Periodic sampling image diagram

Data collected by periodic sampling is retained as the latest diagnostic information if it is the diagnostic information of the node corresponding to 3.2.3.CaoExtension class. For channel data, it is retained as the accumulated data corresponding to the 3.4.3.10.COLLECTDATA <?> variable of the same channel number. The accumulated channel data continues to accumulate until get_value of 3.4.3.10.COLLECTDATA<?> variable is executed, and when get_value is executed, the accumulated data is fetched and the accumulated data is deleted.

For information about how to use this provider, see 4. Programming with WirelessMSCL Providers.

3.2.1.1.3. **Notes on Connection**

When IsBroadcastSetToIdle option is omitted or true is specified during AddController, this provider causes the gateway to execute a command that makes the node idle by broadcasting to the gateway after connecting with the gateway. This command allows the node to transition to the idle state when reconnecting if it disconnects from the gateway unexpectedly.

3.2.1.1.4. **When a disconnection occurs between the gateway and the client PC**

If the connection between the gateway and the client PC is broken for any reason, the same Controller cannot be acquired, for example, even if the gateway and the client PC are connected again. Delete the object of the target Controller class and recreate Controller class.

3.2.2. CaoController classes

3.2.2.1. **GetVariableNames method**

Gets a list of variable names that can be connected.

Format

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

Sample usage (C#)

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

3.2.2.2. **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];
```

3.2.2.3. **AddExtension method**

Add the extension board object corresponding to the node to CaoController. When adding an object, it will

succeed even if the connection with the node fails. Connecting to a node is accomplished by executing CaoExtension class Execute method or by acquiring/setting CaoExtension class variables @COMMPROTOCOL, @SERIALNO, @CLOUD_NAME, @FIRMWARE_VERSION, and @MODEL, @SAMPLING. The status of the node can be checked with the @ LASTSTATUS.

Please refer to 5. Appendix B. Notes and Actions for Connecting to Nodes for precautions when connecting to nodes.0

The following are the specifics of AddExtension method:

Format

```
AddExtension
(
"<expansion board name>",                    // Expansion Board Name (Optional)
"<Option>"                  // Option string
)
```

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
AddressNo	✓	Specifies the address number of the target node. Only one node with the same address number can be added to the same gateway.	1 -	-----
Sampling	-	Specifies the sampling period for the node. Specify the specifiable values from Table 3-2.	See Table 3-2	Uses the sampling period set for the device.

※ The sampling period can only be set for idle nodes.

Sample usage (C#)

```
// Adding an expansion board
ORiN2.ManagedCAO.CCaoExtension caoExt = controller.AddExtension("Node4228",
                                                    "AddressNo=4228,Sampling=47");
```

Table 3-2 List of sampling intervals for the specifiable nodes

Value	Meaning
46	300Hz
47	800Hz
48	1,600Hz
49	3,200Hz
55	12,500Hz
56	25,000Hz
57	62,500Hz
58	78,125Hz
60	104,170Hz
62	1kHz
63	2kHz
64	3kHz
65	4kHz
66	5kHz
67	6kHz
68	7kHz
69	8kHz
70	9kHz
71	10kHz
72	20kHz
73	30kHz
74	40kHz
75	50kHz
76	60kHz
77	70kHz
78	80kHz
79	90kHz
80	100kHz
98	887Hz

Value	Meaning
100	8,192Hz
101	4,096Hz
102	2,048Hz
103	1,024Hz
104	512Hz
105	256Hz
106	128Hz
107	64Hz
108	32Hz
109	16Hz
110	8Hz
111	4Hz
112	2Hz
113	1Hz
114	2sec
115	5sec
116	10sec
117	30sec
118	1min
119	2min
120	5min
121	10min
122	30min
123	60min
127	24hours

3.2.2.4. **AddVariable method**

Adds a variable object to CaoController. Only the variable names shown in 3.4.2 can be used.

AddVariable is specifi ed as follows.

Format

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

3.2.2.5. **Execute method**

Execute CaoController extension command. Only the ones shown in 0 can be used in the command name.0 Execute is specifi ed as follows.

Format

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

3.2.2.6. **OnMessage event**

You can receive controller error notifications and status changes as OnMessage events. See 3.5 for the events that can be received.

3.2.3. CaoExtension classes

This is an extension board object corresponding to the node. When an object of this class is deleted, SetToIdle command is executed for the target node, and the idle state is changed from the sampled state to the idle state.

3.2.3.1. **GetVariableNames method**

Gets a list of variable names that can be connected.

Format

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

Sample usage (C#)

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

3.2.3.2. **Variables Properties**

Gets the variable collection held by the expansion board.

Sample usage (C#)

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

3.2.3.3. **AddVariable method**

Adds a variable object to CaoExtension. Only the variable names shown in 3.4.3 can be used.

AddVariable is specifi ed as follows.

Format

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

3.2.3.4. **Execute method**

Execute CaoExtension extension command. Only the command names shown in 3.3.2 can be used.

Execute is specifi ed as follows.

Format

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

3.2.4. CaoVariable classes

3.2.4.1. **Value Properties**

Gets/sets data from the connected gateway or node. The behavior depends on the variable name. For details, refer to section 3.4, Variable List.

3.3. Extended command list

Defines the extended commands available for each class.

3.3.1. CaoController class-command

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

Command	Description	Reference
BroadcastSettoIdle	Idle the node by broadcast.	P.22
SyncStartSampling	Starts synchronous sampling.	P.23

3.3.1.1. **BroadcastSetToIdle Commands**

Idle all nodes in the same frequency band as the gateway by broadcast. This command continues to acquire the status until the node of the existing Extension is idle for the timeout period specified at the time of AddController, and returns the status. If all of the nodes in the existing Extension are idle, it succeeds. Returns the status after the timeout period, even if the node is not idle.

Immediately after execution, the physical node state is changed, but the status of the return value may not change. In this case, it may be reflected in the status after a short time. Obtain the @LASTSTATUS and check the status of the node again.

The following are the arguments and return values:

Data type

Item	Type Description	
Argument	None	
Return value	VT_ARRAY | VT_UI1	
	i	The status is stored in ascending order of the node address number of the existing Extension class. For more information on the status, see Data Types in 3.4.3.8.

※i : Number of CaoExtension classes added to CaoContoroller

※If CaoExtension does not exist, VT_EMPTY is returned.

Sample usage (C#)

```
// Execute BroadcastSetToIdle
Object nodeStatus = controller.Execute("BroadcastSetToIdle", null) as object;

// The result of the command
If (nodeStatus != null)
{
    If (nodeStatus is ushort[])
```

```
    {
        Ushort[] status = nodeStatus as ushort[];
        For (int i = 0; i < status.Length; i++)
        {
            // Status of each node (address number ascending order)
            Ushort state = status[i];
        }
    }
}
```

3.3.1.2. **SyncStartSampling Commands**

Starts synchronous sampling of the gateway using the node-information of Extension that it holds. This command continues to acquire the status until the node of the existing Extension is idle for the timeout period specified at the time of AddController, and returns the status. If all of the nodes in the existing Extension are idle, it succeeds. Returns the status after the timeout period, even if the node is not idle.

To end synchronous sampling, the node must be idle from the sampling state. To make the node idle, run BroadcastSetToIdle command for Controller class or SetToIdle command for Extension class.

The following are the arguments and return values:

Data type

Item	Type Description	
Argument	None	
Return value	VT_ARRAY | VT_UI1	
	i	The status is stored in ascending order of the node address number of the existing Extension class. For more information on the status, see Data Types in 3.4.3.8.

※i : Number of CaoExtension classes added to CaoController

※If CaoExtension does not exist, VT_EMPTY is returned.

Sample usage (C#)

```
// Execute SyncStartSampling
Object resultStatus = controller.Execute("SyncStartSampling", null) as object;
// Node status after command execution
If (resultStatus != null)
{
    If (resultStatus is ushort[])
    {
        Ushort[] status = resultStatus as ushort[];
```

```
            For (int i = 0; i < status.Length; i++)
            {
                // Status of each node (address number ascending order)
                Ushort state = status[i];
            }
        }
    }
```

3.3.2. CaoExtension class-command

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

Command	Description	Reference
SettoIdle	Idle the node.	P.24
Ping	Confirm the connection with the node.	P.25
GetActiveChannels	Gets the active channel for the current node.	P.25
EntrySampling	Joins a group of synchronous samples.	P.26

3.3.2.1. **SetToIdle Commands**

Sample or sleep the node to idle. Check the status of the node until it is idle for the specified timeout period during AddController.

If the node is not idle at the time of AddExtension, this command cannot make the node idle. Be sure to leave the node idle at the time of AddExtension. If the node is idle at the time of AddExtension, this command can change the node to idle state.

The following are the arguments and return values:

Data type

Item	Type Description	
Argument	None	
Return value	VT_UI1	The status of the node after the command is executed. Refer to 3.4.3.8 for more information on the status.

Sample usage (C#)

```
    // Execute SetToIdle
    Ushort? status = extension.Execute("SetToIdle", "") as ushort?;
```

3.3.2.2. **Ping command**

Check whether the node can communicate. Success only if the node is idle. The return values are shown below.

Data type

Item	Type Description	
Argument	None	
Return value	VT_BOOL	Success or failure of communication True : Communication enabled False : Communication disabled

Sample usage (C#)

```
// Execute Ping
Bool? result = extension.Execute("Ping", "") as bool?;
```

3.3.2.3. **GetActiveChannels Commands**

Gets a list of valid channel numbers for a node. The acquired channel number can be specified with the 3.4.3.10. COLLECTDATA<?> variable option to obtain the target channel data.

The following are the arguments and return values:

Data type

Item	Type Description	
Argument	None	
Return value	VT_ARRAY | VT_UI1	
	i	Valid channel number

※ i : Number of valid channels

※ If a valid channel does not exist, VT_EMPTY is the return value.

Sample usage (C#)

```
// Execute GetActiveChannels
Object activeChannels = extension.Execute("GetActiveChannels", "") as object;
// Check for a valid channel
If (activeChannels != null)
{
    If (activeChannels is byte[])
    {
        Byte[] channels = activeChannels as byte[];
        For (int i = 0; i < channels.Length; i++)
        {
```

```
                // Valid channel number
                Byte activeChannel = channels[i];
            }
        }
    }
```

3.3.2.4. **EntrySampling Commands**

Join the node to a group of synchronous samples. If you join a group with synchronous sampling, you can start sampling of the target node at the time of synchronous sampling. When AddExtension method is executed, the node joins the group with synchronous sampling. However, if the connection with the node fails, the group for synchronous sampling goes into an unjoined state. If the node is placed in the idle state after executing synchronous sampling, re-execute this command. Refer to 3.3.2.4.1 for the group of synchronous sampling.

The following are the arguments and return values:

Data type

Item	Type Description	
Argument	None	
Return value	VT_BOOL	Participation status True : Joined False : Not joined

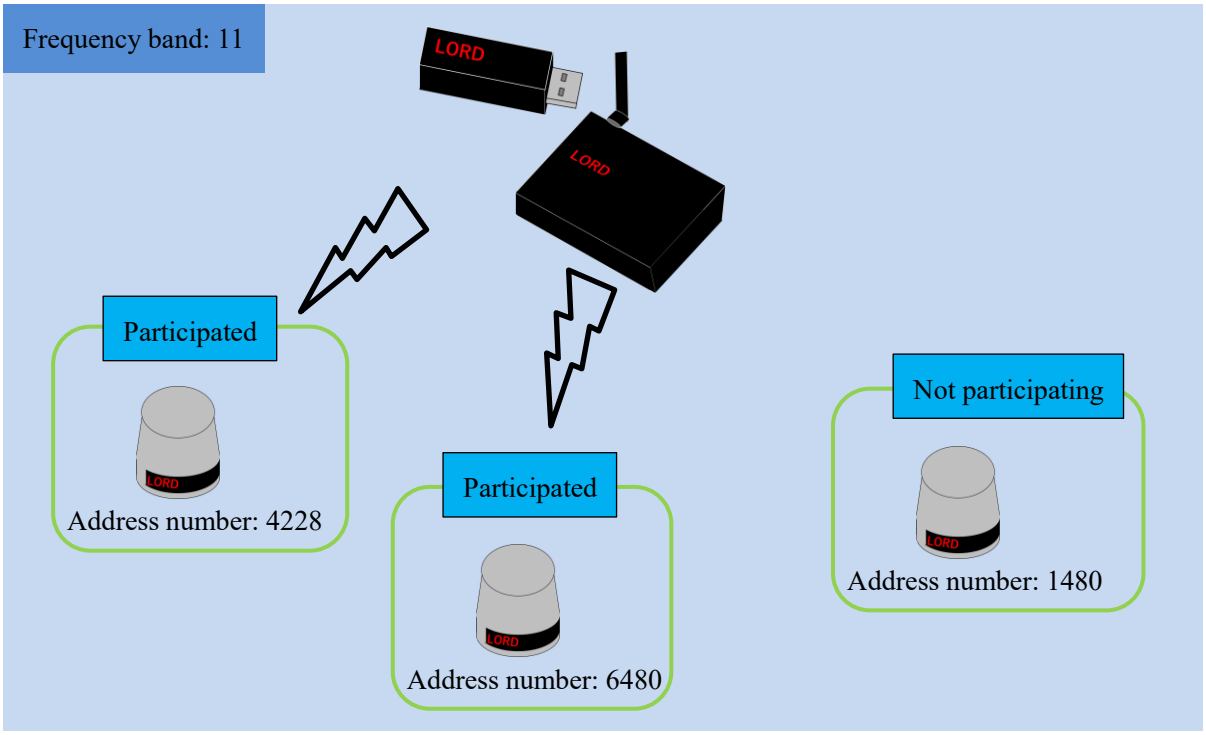
Sample usage (C#)

```
// Execute EntrySampling
Bool? entry = extension.Execute("Entrysampling", "") as bool?;
```

3.3.2.4.1. **About Synchronous Sampling Groups**

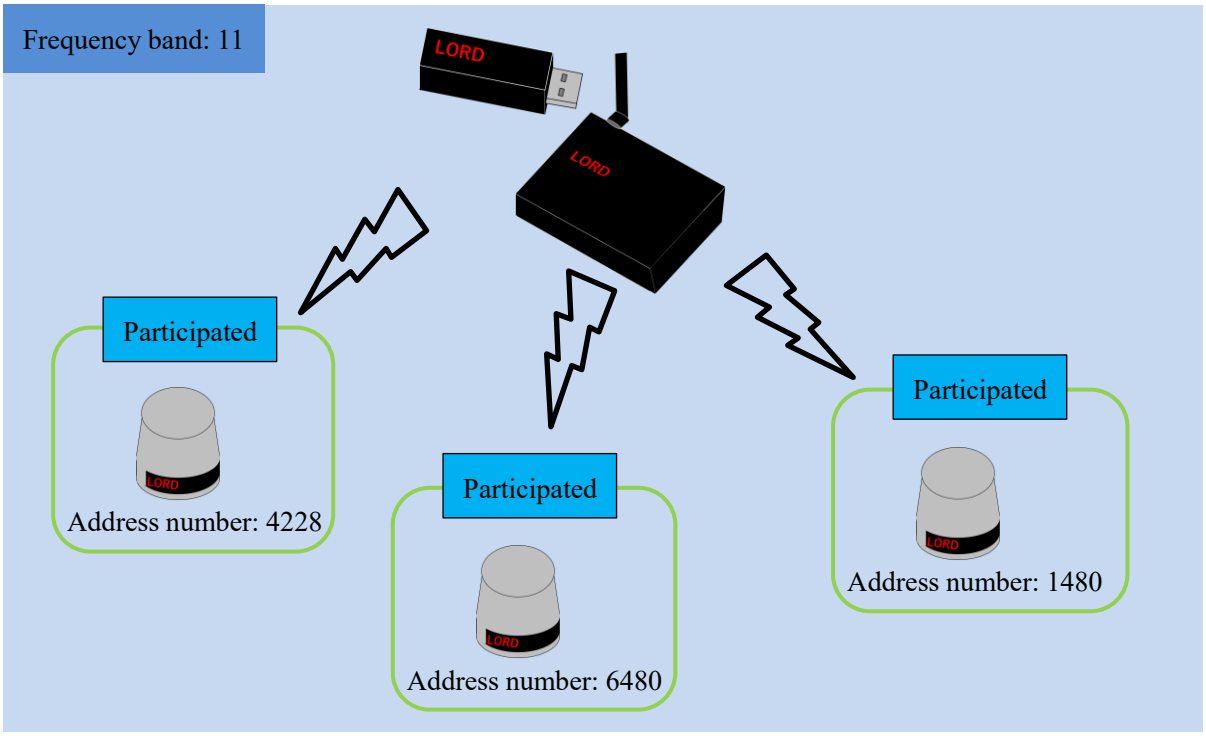
The following is a detailed description of the group for synchronous sampling. This section explains when synchronous sampling is started when the group of synchronous sampling is not joined, or when this command is executed to start synchronous sampling after joining the group of synchronous sampling. In this example, a node with node address number 1480 is joined to the group of synchronous sampling.

- **Node 1480 is not participating in a group of synchronous samplings**



When Controller class-based SyncStartSampling command is executed as shown in the above figure, synchronous sampling is started using the node with node address number 4228 and node address number 6480. Nodes with node address number 1480 are not included in synchronous sampling.

・**Node 1480 joins a group with synchronous sampling**



Join the synchronous sampling group using node address number 1480 as EntrySampling command. If you execute SyncStartSampling command of Controller class after joining in the status shown in the above figure, synchronous sampling is started using the node with node address number 4228, 6480, 1480.

3.4. Variable list

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

3.4.1. System and User Variables

There are two types of variables in the provider: system variables and user variables.

System Variables

A variable that accesses only the information in the object that holds the variable. System variables are often static data. System variables are preceded by "@".

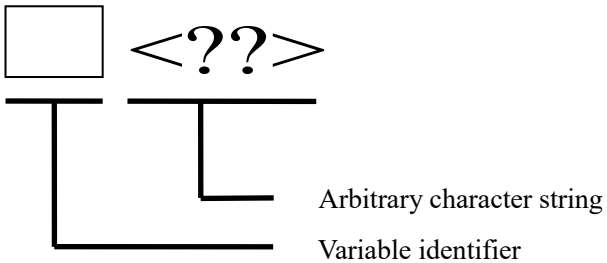
e.g. provider version, device manufacturer, serial number

User variable

A variable that you can use to specify what information you want to access when you create a variable by using an option string. Due to its nature, user variables can be given arbitrary strings after variable identification in order to register multiple variables (useful if you only want to change options, etc.).

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

Multi-variable common specification format



3.4.2. CaoController class-variable

Variable Name	Description	Value		Reference
		Get	Put	
@MAKER_NAME	Obtain the manufacturer's name.	✓	-	P.30
@VERSION	Get the provider version.	✓	-	P.30
@NETWORKCAPACITY	Gets the current network capacity used for synchronous sampling.	✓	-	P.30
@COMMPROTOCOL	Gets the gateway communication protocol.	✓	-	P.31
@SERIALNO	Get the gateway serial number.	✓	-	P.31
@CLOUD_NAME	Get the Universal Gateway name to use for SenserCloud.	✓	-	P.32

@FIRMWARE_VERSION	Gets the firmware version of the gateway.	✓	-	P.32
@MODEL	Gets the model of the gateway.	✓	-	P.32

3.4.2.1. **@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;
```

3.4.2.2. **@VERSION**

Gets the version of EXE.

Data type

Type Description	
VT_BSTR	Get the version of EXE. *.*.*

.

Sample usage (C#)

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

3.4.2.3. **@NETWORKCAPACITY**

Gets the current network usage as a percentage when performing synchronous sampling. Gets the network capacity based on the information of the node corresponding to CaoExtension class held by CaoController class. If this value is greater than 100%, SyncStartSampling command cannot be executed. To change the amount of network space, you must specify the value of Sampling option during AddExtension or change the value in the @SAMPLING variable of Extension class to change the sampling period. The number of nodes and the number of valid channels of nodes corresponding to CaoExtension class to be retained also affect.

Data type

Type Description	
VT_R4	Get the current network usage.

Sample usage (C#)

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

3.4.2.4. @COMMPROTOCOL

Gets the gateway communication protocol.

Data type

Type Description	
VT_UI1	Gets the communication protocol. The following values are retrieved: 0 : LXRS 1 : LXRS+

Sample usage (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@COMMPROTOCOL", "");
// Acquisition of Values
Byte? commProtocol = var.Value as Byte?;
```

3.4.2.5. @SERIALNO

Get the gateway serial number.

Data type

Type Description	
VT_BSTR	Get the serial number.

Sample usage (C#)

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

3.4.2.6. @CLOUD_NAME

Obtain the universal base station name for use with Parker's SensorCloud cloud application.

Data type

Type Description	
VT_BSTR	Gets the Universal Base Station name.

Sample usage (C#)

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

3.4.2.7. @FIRMWARE_VERSION

Gets the firmware version of the gateway.

Data type

Type Description	
VT_BSTR	Gets the firmware version. *.*.*

Sample usage (C#)

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

3.4.2.8. @MODEL

Get the gateway model.

Data type

Type Description	
VT_BSTR	Get model. See MSCL documentation for more information about the values.

Sample usage (C#)

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

3.4.3. CaoExtension class-variable

Variable Name	Description	Value		Reference
		Get	Put	
@COMMPROTOCOL	Gets the communication protocol of the node.	✓	-	P.33
@SERIALNO	Get the serial number of the node.	✓	-	P.34
@CLOUD_NAME	Gets the universal sensor name for use with SenserCloud.	✓	-	P.34
@FIRMWARE_VERSION	Gets the firmware version of the node.	✓	-	P.34
@MODEL	Gets the model of the node.	✓	-	P.35
@SAMPLING	Acquires/sets the sampling period of the node.	✓	✓	P.35
@DIAGNOSTIC_INFO	Gets diagnostic information sent from a node.	✓	-	P.36
@LASTSTATUS	Get the last status of the node.	✓	-	P.38
@ENTRY	Gets the group join status of synchronous sampling for a node.	✓	-	P.38
COLLECTDATA<??>	Gets the channel data.	✓	-	P.39

※ If the connection to the node succeeds at least once, the value at the time of the connection is retrieved.

The availability of acquisition varies depending on the state of the node. The executable state of each state of the node is described below.

Variable Name	Availability		
	Sleep State	Idle state	Sampling state
@COMMPROTOCOL	×	✓	✓
@SERIALNO	×	✓	✓
@CLOUD_NAME	✓	✓	✓
@FIRMWARE_VERSION	×	✓	✓
@MODEL	×	✓	✓
@SAMPLING	×	✓	×
@DIAGNOSTIC_INFO	✓	✓	✓
@LASTSTATUS	✓	✓	✓
COLLECTDATA<??>	✓	✓	✓

3.4.3.1. **@COMMPROTOCOL**

Gets the communication protocol of the node.

Data type
Type Description

VT_UI1	Gets the communication protocol. The following values are retrieved: 0 : LXRS 1 : LXRS+

Sample usage (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = extension.AddVariable("@COMMPROTOCOL", "");
// Acquisition of Values
Byte? value= var.Value as Byte?;
```

3.4.3.2. @SERIALNO

Get the serial number of the node.

Data type

Type Description	
VT_BSTR	Get the serial number.

Sample usage (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = extension.AddVariable("@SERIALNO", "");
// Acquisition of Values
String serialNo = var.Value as string;
```

3.4.3.3. @CLOUD_NAME

Obtain the universal sensor name for use with Parker's SensorCloud cloud application.

Data type

Type Description	
VT_BSTR	Gets the universal sensor name.

Sample usage (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = extension.AddVariable("@CLOUD_NAME", "");
// Acquisition of Values
String cloudName = var.Value as string;
```

3.4.3.4. @FIRMWARE_VERSION

Gets the firmware version of the node.

Data type

Type Description	
VT_BSTR	Gets the firmware version. *.*.*

Sample usage (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = extension.AddVariable("@FIRMWARE_VERSION", "");
// Acquisition of Values
String firmwareVersion = var.Value as string;
```

3.4.3.5. **@MODEL**

Get the model of the node.

Data type

Type Description	
VT_BSTR	Get model. See MSCL documentation for more information about the values.

Sample usage (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = extension.AddVariable("@MODEL", "");
// Acquisition of Values
String model = var.Value as string;
```

3.4.3.6. **@SAMPLING**

Acquires/sets the sampling period of the node during synchronous sampling. The actual sampling period is reflected to the node when the 3.3.1.2. SyncStartSampling command is executed.3.3.1.2

Data type

Type Description	
VT_I4	Sampling period. See Table 3-2 for details on the values.

Sample usage (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable varSampling = extension.AddVariable("@SAMPLING", "");
// Acquisition of Values
Int32? sampling = varSampling.Value as Int32?;


// Set value
```

```
VarSampling.Value = 105;
```

3.4.3.7. **@DIAGNOSTIC_INFO**

If the node is idle, diagnostic information is periodically sent. WirelessMSCL providers retain only the most recent diagnostic information sent, and this variable retrieves the diagnostic information held.

Data type

Type Description		
VT_ARRAY | VT_VARIANT		Diagnostic information
0	VT_UI1	Current node state
1	VT_UI4	Idle execution time (seconds)
2	VT_UI4	Sleep execution time (seconds)
3	VT_UI4	Active execution time (seconds)
4	VT_UI4	Inactive execution time (seconds)
5	VT_UI2	Reset counter
6	VT_UI4	Embedded test results
7	VT_R4	Internal temperature (℃)
8	VT_UI1	Low Battery Indicator
9	VT_UI1	External power supply
10	VT_UI4	Sweep index
11	VT_UI4	Bad sweeps
12	VT_UI4	Total number of transmissions
13	VT_UI4	Total number of retransmissions
14	VT_UI4	Total number of dropped packets
15	VT_UI4	Number of Synchronization Attempts
16	VT_UI4	Number of Synchronization Failures
17	VT_UI4	Time elapsed since last synchronization (seconds)
18	VT_UI4	Event Trigger Index
19	VT_R4	Data logging memory status (%)

※ If there is no corresponding item in the transmitted data, VT_EMPTY is returned.

Sample usage (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = extension.AddVariable("@DIAGNOSTIC_INFO", "");
// Acquisition of Values
Object[] diagnosticInfo = var.Value as object[];
```

```
If (diagnosticInfo != null)
{
    // Current node state
    Byte? status = diagnosticInfo[0] as Byte?;
    // Idle execution time (seconds)
    UInt32? idleRuntime = diagnosticInfo[1] as UInt32?;
    // Sleep execution time (seconds)
    UInt32? sleepRuntime = diagnosticInfo[2] as UInt32?;
    // Active execution time (seconds)
    UInt32? activeRuntime = diagnosticInfo[3] as UInt32?;
    // Inactive execution time (seconds)
    UInt32? nonActiveRuntime = diagnosticInfo[4] as UInt32?;
    // Reset counter
    UInt16? resetCounter = diagnosticInfo[5] as UInt16?;
    // Embedded test results
    UInt32? testResult = diagnosticInfo[6] as UInt32?;
    // Internal temperature (°C)
    Float? inTemp = diagnosticInfo[7] as float?
    // Low Battery Indicator
    Byte? lowBattery = diagnosticInfo[8] as Byte?;
    // External power supply
    Byte? extrnalPower = diagnosticInfo[9] as Byte?;
    // Sweep index
    UInt32? sweepIndex = diagnosticInfo[10] as UInt32?;
    // Bad sweeps
    UInt32? badSweepCount = diagnosticInfo[11] as UInt32?;
    // Total number of transmissions
    UInt32? totalSendCount = diagnosticInfo[12] as UInt32?;
    // Total number of retransmissions
    UInt32? totalReSendCount = diagnosticInfo[13] as UInt32?;
    // Total number of dropped packets
    UInt32? totalDropPacketCount = diagnosticInfo[14] as UInt32?;
    // Number of Synchronization Attempts
    UInt32? syncTryCount = diagnosticInfo[15] as UInt32?;
    // Number of Synchronization Failures
    UInt32? syncFaildCount = diagnosticInfo[16] as UInt32?;
```

```
        // Time elapsed since last synchronization (seconds)
        UInt32? lastSyncTime = diagnosticInfo[17] as UInt32?;
        // Event trigger index (seconds)
        UInt32? eventTriggerIndex = diagnosticInfo[18] as UInt32?;
        // Data logging memory status (%)
        Float? memmory = diagnosticInfo[19] as float?;
    }
```

3.4.3.8. **@LASTSTATUS**

Gets the status of the last connection to the node.

Data type

Type Description	
VT_UI1	The status of the node. Details of the status are as follows. 0 : Idle state 1 : Sleep State 2 : Sampling state 3 : Sampling status, but losing beacon 4 : Sampling state, but not activity (in activity detection mode) 255 : Unknown

Sample usage (C#)

```
    // Add Variable
    ORiN2.ManagedCAO.CCaoVariable var = extension.AddVariable("@LASTSTATUS", "");
    // Acquisition of Values
    Int? lastStatus = var.Value as int?;
```

3.4.3.9. **@ENTRY**

Gets the join status of the node to the synchronous sampling group. If this value is true, executing SyncStartSampling command of Controller class starts synchronous sampling of the node. If false, synchronous sampling is not started. You can join the synchronous sampling group by executing EntrySampling command of Extension class.

Data type

Type Description

VT_BOOL	Participation in the synchronous sampling group. True : Joined False : Not joined

Sample usage (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = extension.AddVariable("@ENTRY", "");
// Acquisition of Values
Bool? entry = var.Value as bool?;
```

3.4.3.10. **COLLECTDATA<??>**

Retrieves the channel data accumulated by periodic sampling. Enter any character string after COLLECTDATA to specify the variable name.

Option

Option	Required	Description	Value Range	Default value
ChannelNo	✓	Specifies the channel number of the channel data.	1 - 16	-----

Data type

Type Description			
VT_ARRAY | VT_VARIANT			Stored data
0	VT_ARRAY | VT_BSTR		Accumulated timestamp
	I	VT_BSTR	Timestamp corresponding to each channel data
1	VT_ARRAY | VT_VARIANT		Accumulated channel data
	I	VT_VARIANT	The data that is retrieved depends on the data type of the channel data.

※ If there is no accumulated data, VT_EMPTY is returned.

※ i : Amount of data accumulated

Sample usage (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = extension.AddVariable("COLLECTDATA1", "ChannelNo=1");
// Acquisition of Values
Object collectValues = var.Value as object;

// Value decomposition
```

```
If (collectValues != null)
{
    Object[] values = collectValues as object[];
    // Timestamp portion
    String[] timestamps = values[0] as string[];

    For (int i = 0; i < timestamps.Length; i++)
    {
        String timestamp = timestamps[i];
    }

    // Accumulated data portion
    Object[] collectDatas = values[1] as object[];

    For (int i = 0; i < collectDatas.Length; i++)
    {
        Object collectData = collectDatas[i];
    }
}
```

3.4.3.10.1. **About Channel Data Types**

The acquired channel data is stored as the following data.

- **If the datatype is float**

Data type

VT_R4

- **If the datatype is double**

Data type

VT_R8

- **If the datatype is uint8**

Data type

VT_UI1

- **If the datatype is uint16**

Data type

VT_UI2

- **If the datatype is uint32**

Data type

VT_UI4

・**If the datatype is int8**

Data type

VT_I1

・**If the datatype is int8**

Data type

VT_I1

・**If the datatype is int16**

Data type

VT_I2

・**If the datatype is int32**

Data type

VT_I4

・**If the datatype is bool**

Data type

VT_BOOL

・**If the datatype is float**

Data type

VT_R4

・**If the datatype is string or Timestamp**

Data type

VT_BSTR

・**If the datatype is Bytes**

Data type

VT_ARRAY | VT_UI1

・**When the data type is Vector (data in a one-dimensional array)**

Data type

VT_ARRAY | VT_R4

Or

VT_ARRAY | VT_R8

Or

VT_ARRAY | VT_UI2

Or

VT_ARRAY | VT_UI1

・**When the data type is Matrix (data in a two-dimensional array)**

Data type

VT_ARRAY | VT_VARIANT

i	VT_ARRAY | VT_R4 Or VT_ARRAY | VT_R8 Or VT_ARRAY | VT_UI2 Or VT_ARRAY | VT_UI1

※ i : Depends on the data to be acquired.

・**If the datatype is ChannelMask**

Data type

VT_ARRAY | VT_UI4	
i	Valid channel number

※ i : Number of valid channels

・**If the datatype is RfSweep**

Data type

VT_ARRAY | VT_VARIANT			
i	VT_ARRAY | VT_VARIANT		
	0	VT_UI4	Key
	1	VT_I2	Value

※ i : Number of combinations

・**If the datatype is StructuralHealth**

Data type

VT_ARRAY | VT_VARIANT				
0	VT_R4			Angle being measured
1	VT_UI4			Uptime counter
2	VT_R4			Damage that occurred (0% = no damage, 100% = dead)
3	VT_ARRAY | VT_UI4			Sample Rate
	0	Sample Rate Type		
	1	Number of sample rates		
4	VT_ARRAY | VT_VARAINT			Histogram
	i	VT_ARRAY | VT_VARIANT		Bin
		0	VT_VARIANT	Starting range of the bin
		1	VT_VARIANT	Bin End Range

		2	VT_VARIANT	Number of values within the range of the bin

※ i : Number of combinations

3.5. Event list

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

No.	Description
0	Data memory capacity over message

3.5.1.1. **Data memory capacity over message**

This event occurs when the amount of memory used by the provider exceeds the threshold when data is acquired by periodic sampling. When this event occurs, the old stored data is output, and the newly acquired data is stored. This event occurs when the accumulated speed during periodic sampling is faster than the speed at which COLLECTDATA<?> variable is acquired in 3.4.3.10.

To prevent this event from occurring, slow down the speed stored in the gateway, or speed up the acquisition of 3.4.3.10. COLLECTDATA<?> variables, or both.

Value Property Data Types

Type Description		
VT_ARRAY | VT_VARIANT		
0	VT_UI4	Node address number
1	VT_UI1	Channel number
2	VT_ARRAY | VTVARIANT	Old accumulated channel data. For details of the data, see 3.4.3.10.

Sample usage (C#)

```
///<summary>
/// Processing method for message reception
///</summary>
///<param name="errorData><erroneous data</ param>
Private void OnReceivedError(object[] errorData)
{
    Int32? errorCode = errorData[0] as Int32?;
    String packet = errorData[1] as String;
}
```

4. Programming by WirelessMSCL providers

With WirelessMSCL providers, you can connect and sample client PCs and gateways by following these steps.

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

After connecting to the gateway, the node information can be accessed by generating CaoController's CaoExtension object, or the gateway information can be accessed by generating CaoVariable object of CaoController.

- Creating a CaoExtension

Node-information can be accessed by creating a CaoVariable of CaoExtension.

4.1. Sample Programming to Start Synchronous Sampling and Get Node Channel Data

This example shows a sample program that starts synchronous sampling and obtains the channel data value of the node from the gateway.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	Link with serial communications
	The host COM port number is 5.
	The node address number is 4228.
	Channel number is 1
Process Description	Adding Channel Data to Retrieve
	Start synchronous sampling of the gateway
	Get the channel data

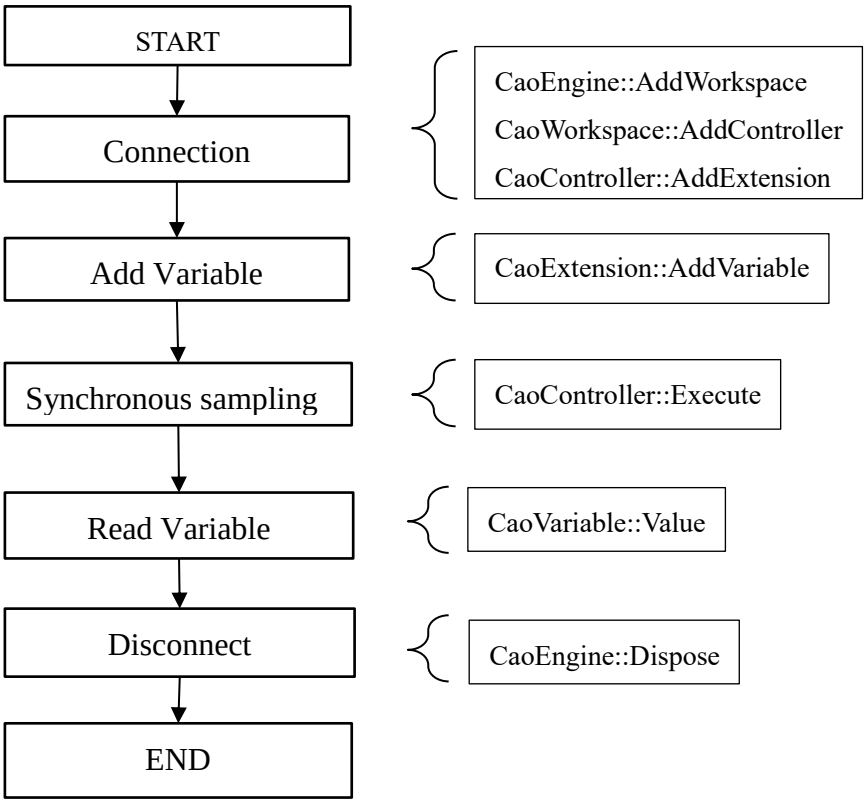


Fig. 4-1 Flow of Starting Synchronous Sampling and Acquiring Node Channel Data

Specific codes are given in the following sections.

4.1.1. Sample program

The following is an overview of the sample program.

Sample	Sample1.cs

```
// Object
Private ORiN2.ManagedCAO.CCaoEngine m_caoEngine = null;
Private ORiN2.ManagedCAO.CCaoWorkspace m_caoWorkspace = null;
Private ORiN2.ManagedCAO.CCaoController m_caoController = null;
Private ORiN2.ManagedCAO.CCaoExtension m_caoExtension = null;
Private ORiN2.ManagedCAO.CCaoVariable m_varChannelData = null;

Public void Main()
{
    // Connection
    This.Connect();

    // Synchronous sampling start
    This.m_caoController.Execute("SyncStartSampling", null);

    // Wait for data to be collected
    System.Threading.Thread.Sleep(40);

    // Acquisition of Values
    Object collectValues = this.m_varChannelData.Value as object;

    // Value decomposition
    If (collectValues != null)
    {
        Object[] values = collectValues as object[];
        // Timestamp portion
        String[] timestamps = values[0] as string[];

        For (int i = 0; i < timestamps.Length; i++)
        {
            String timestamp = timestamps[i];
        }
```

```
            // Accumulated data portion
            Object[] collectDatas = values[1] as object[];

            For (int i = 0; i < collectDatas.Length; i++)
            {
                Object collectData = collectDatas[i];
            }
        }
        // Disconnect
        This.Disconnect();
    }
    ///<summary>
    /// Connection method
    ///</summary>
    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("WirelessGateway",
                                                                 "CaoProv.LORD.WirelessMSCL",
                                                                 "",
                                                                 "Conn=COM:5,DataCollectionCycl
e=30,IsBroadcastSetToIdle=true");
        Creating CaoExtension Objects
        This.m_caoExtension = this.m_caoController.AddExtension("Node4228", "AddressNo=4228");
        // Adding Channel Data to Extension
        This.m_varChannelData = this.m_caoExtension.AddVariable("COLLECTDATA1", "ChannelNo=1");
    }

    ///<summary>
    /// Disconnect method
    ///</summary>
    Private void Disconnect()
```

```
    {
        This.m_caoEngine.Dispose();
        This.m_caoEngine = null;
    }
```

4.1.1.1. **Connection**

To connect to the gateway and nodes, proceed as follows:

(1) Prepare a variable to hold the object.

The objects required to connect to the gateway are CaoEngine objects, CaoWorkspace objects, and CaoController objects.CaoWorkspace object does not need to have a variable to retrieve CaoController object from CaoWorkspaces. It also requires a CaoExtension to connect to the node. You will also need a CaoVariable for accessing the variable. The following is a code example for C#.

Sample usage (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;
    // Variables for CaoExtension
    Private ORiN2.ManagedCAO.CCaoExtension m_caoExtension = null;
    // Variables for CaoVariable
    Private ORiN2.ManagedCAO.CCaoVariable m_varChannelData = null;
```

(2) Creates a CaoEngine.

CaoEngine is generated using the New keyword.

Sample usage (C#)

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

(3) Gets or generates a CaoWorkspace container.

When you create a CaoEngine object, it defaults to generating one CaoWorkspaces object and one CaoWorkspace object. The following is a sample code/default CaoWorkspace for creating a new CaoWorkspace.

Sample usage (C#)

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

(4) Creates a CaoController.

To generate a CaoController object, configure the provider name to use and the parameters to use. For WirelessMSCL providers, optionally specify the COM port number of the connected gateway, the frequency at which WirelessMSCL provider periodically collects data to the gateway, and whether nodes in the same frequency band as the gateway are idle when connected. The following is a code example:

Sample usage (C#)

```
// Generate CaoController object
This.m_caoController = this.m_caoWorkspace.AddController("WirelessGateway",
                                                         "CaoProv.LORD.WirelessMSCL",
                                                         "",
                                                         "Conn=COM:5,DataCollectionCycl
e=30,IsBroadcastSetToIdle=true");
```

(5) Creates a CaoExtension.

Create a CaoExtension for the node you want to connect to. The following is a sample code for generating a Extension object that accesses a node with node address number 4228.

Sample usage (C#)

```
/ Creating CaoExtension Objects
This.m_caoExtension = this.m_caoController.AddExtension("Node4228", "AddressNo=4228");
```

(6) Creates a CaoVariable.

Create a CaoVariable object of the channel data you want to retrieve. When you create a CaoVariable object, WirelessMSCL providers begin accumulating channel data. The following is a code for generating a Variable object that accesses the channel data of channel number 1 of the node created in step (5).

Sample usage (C#)

```
Creating CaoVariable Objects
This.m_varChannelData = this.m_caoExtension.AddVariable("COLLECTDATA1",
 "ChannelNo=1");
```

4.1.1.2. **Starting Synchronous Sampling**

Use the connected gateway and nodes to start synchronous sampling. By starting synchronous sampling, data is sent from the connected node to the gateway. WirelessMSCL provider periodically collects the data accumulated in the gateway in Variable object added in 4.1.1.1 and accumulates the channel data. The following

is a code example:4.1.1.1

Sample usage (C#)

```
// Synchronous sampling start
This.m_caoController.Execute("SyncStartSampling", null);
```

4.1.1.3. **Retrieving Channel Data**

To get the value of accumulated channel data, refer to Value property of CaoVariable object. When acquiring the value of channel data, it is acquired by dividing it into the time stamp part and the accumulated data part. The following is a code example:

Sample usage (C#)

```
// Acquisition of Values
Object collectValues = this.m_varChannelData.Value as object;

// Value decomposition
If (collectValues != null)
{
    Object[] values = collectValues as object[];
    // Timestamp portion
    String[] timestamps = values[0] as string[];

    For (int i = 0; i < timestamps.Length; i++)
    {
        String timestamp = timestamps[i];
    }

    // Accumulated data portion
    Object[] collectDatas = values[1] as object[];

    For (int i = 0; i < collectDatas.Length; i++)
    {
        Object collectData = collectDatas[i];
    }
}
```

4.1.1.4. **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:

Sample usage (C#)

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

4.2. Sample programming to obtain the network capacity of synchronous sampling from the gateway

This example shows a sample program that obtains the value of the network capacity for synchronous sampling from the gateway.Table 4-2 describes the requirements of the sample program, and Fig. 4-2 describes the flow of the sample program.

Table 4-2 Sample program requirements

Requirements	Description
Host	Link with serial communications
	The host COM port number is 5.
	The node address numbers are 4228 and 6480.
Process Description	Get network capacity after connection
	Sets the sampling period for each node.
	Retrieve network capacity again

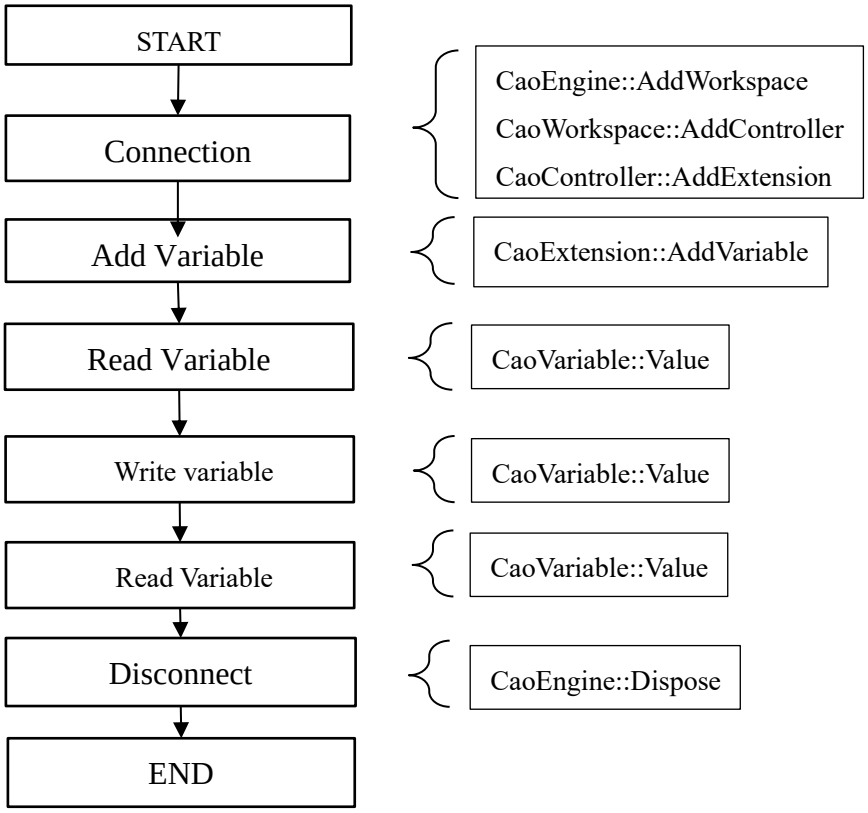


Fig. 4-2 Flow of acquiring the network capacity for synchronous sampling from the gateway

Specific codes are given in the following sections.

4.2.1. Sample program

The following is an overview of the sample program.

Sample	Sample1.cs

```
// Object
Private ORiN2.ManagedCAO.CCaoEngine m_caoEngine = null;
Private ORiN2.ManagedCAO.CCaoWorkspace m_caoWorkspace = null;
Private ORiN2.ManagedCAO.CCaoController m_caoController = null;
Private ORiN2.ManagedCAO.CCaoExtension m_extNode4228 = null;
Private ORiN2.ManagedCAO.CCaoExtension m_extNode6480 = null;
Private ORiN2.ManagedCAO.CCaoVariable m_varNetworkCapacity = null;
Private ORiN2.ManagedCAO.CCaoVariable m_varSampling1 = null;
Private ORiN2.ManagedCAO.CCaoVariable m_varSampling2 = null;

Public void Main()
{
    // Connection
    This.Connect();

    // Get network capacity
    Float? value = this.m_varNetworkCapacity.Value as float?;

    // Change the sampling interval of node 4228 to 2-second intervals.
    This.m_varSampling1.Value = 114;

    // Change the sampling period of node 6480 to 256Hz cycle
    This.m_varSampling2.Value = 105;

    // Retrieve network capacity
    Value = this.m_varNetworkCapacity.Value as float?;

    // Disconnect
    This.Disconnect();
}

///<summary>
/// Connection method
```

```
///</summary>
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("WirelessGateway",
                                                             "CaoProv.LORD.WirelessMSCL",
                                                             "",
                        "Conn=COM:5,DataCollectionCycle=30,IsBroadcastSetToIdle=true");
    // Creating CaoExtension Objects for Node 4228
    This.m_extNode4228 = this.m_caoController.AddExtension("Node4228", "AddressNo=4228");
    // Creating CaoExtension Objects for Node 6480
    This.m_extNode6480 = this.m_caoController.AddExtension("Node6480", "AddressNo=6480");
    // Add Network Capacity Variable
    This.m_varNetworkCapacity = this.m_caoController.AddVariable("@NETWORKCAPACITY", null);
    // Creating a CaoVariable corresponding to the sampling period of node address number 4228
    This.m_varSampling1 = this.m_extNode4228.AddVariable("@SAMPLING", null);
    // Creating a CaoVariable corresponding to the sampling period of node address number 6480
    This.m_varSampling2 = this.m_extNode6480.AddVariable("@SAMPLING", null);
}

///<summary>
/// Disconnect method
///</summary>
Private void Disconnect()
{
    This.m_caoEngine.Dispose();
    This.m_caoEngine = null;
}
```

4.2.1.1. **Connection**

To connect to the gateway and nodes, proceed as follows:

(1) Prepare a variable to hold the object.

The objects required to connect to the gateway are CaoEngine objects, CaoWorkspace objects, and CaoController objects.CaoWorkspace object does not need to have a variable to retrieve CaoController object from CaoWorkspaces. It also requires a CaoExtension to connect to the node. You will also need a CaoVariable for accessing the variable. The following is a code example for C#.

Sample usage (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;
// Variables for CaoExtension
Private ORiN2.ManagedCAO.CCaoExtension m_extNode4228 = null;
Private ORiN2.ManagedCAO.CCaoExtension m_extNode6480 = null;
// Variables for CaoVariable
Private ORiN2.ManagedCAO.CCaoVariable m_varNetworkCapacity = null;
Private ORiN2.ManagedCAO.CCaoVariable m_varSampling1 = null;
Private ORiN2.ManagedCAO.CCaoVariable m_varSampling2 = null;
```

(2) Creates a CaoEngine.

CaoEngine is generated using the New keyword.

Sample usage (C#)

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

(3) Gets or generates a CaoWorkspace container.

When you create a CaoEngine object, it defaults to generating one CaoWorkspaces object and one CaoWorkspace object. The following is a sample code/default CaoWorkspace for creating a new CaoWorkspace.

Sample usage (C#)

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

(4) Creates a CaoController.

To generate a CaoController object, configure the provider name to use and the parameters to use. For

WirelessMSCL providers, optionally specify the COM port number of the connected gateway, the frequency at which WirelessMSCL provider periodically collects data to the gateway, and whether nodes in the same frequency band as the gateway are idle when connected. The following is a code example:

Sample usage (C#)

```
// Generate CaoController object
This.m_caoController = this.m_caoWorkspace.AddController("WirelessGateway",
                                                         "CaoProv.LORD.WirelessMSCL",
                                                         "",
                                                         "Conn=COM:5,DataCollectionCycl
e=30,IsBroadcastSetToIdle=true");
```

(5) Creates a CaoExtension.

Create a CaoExtension for the node you want to connect to. The following shows a sample code for generating a Extension object that accesses a node with node address number 4228 and node address number 6480.

Sample usage (C#)

```
// Creating CaoExtension Objects for Node 4228
This.m_extNode4228 = this.m_caoController.AddExtension("Node4228", "AddressNo=4228");
// Creating CaoExtension Objects for Node 6480
This.m_extNode6480 = this.m_caoController.AddExtension("Node6480", "AddressNo=6480");
```

(6) Creates a CaoVariable.

Create a CaoVariable that sets the amount of network space to retrieve and the sampling period for each node. The following code shows how to create a Variable object that accesses the network capacity and sampling period.

Sample usage (C#)

```
// Creating Network-Space CaoVariable Objects
This.m_varNetworkCapacity = this.m_caoController.AddVariable("@NETWORKCAPACITY", null);
// Creating a CaoVariable corresponding to the sampling period of node address number 4228
This.m_varSampling1 = this.m_extNode4228.AddVariable("@SAMPLING", null);
// Creating a CaoVariable corresponding to the sampling period of node address number 6480
This.m_varSampling2 = this.m_extNode6480.AddVariable("@SAMPLING", null);
```

4.2.1.2. **Obtaining Network Capacity After Connection**

Gets the network capacity for synchronous sampling of connected gateways and nodes. The following is a code example:

Sample usage (C#)

```
// Get network capacity
Float? value = this.m_varNetworkCapacity.Value as float?;
```

4.2.1.3. **Sets the sampling period for each node.**

Sets the sampling period when synchronous sampling is executed for each node. This sampling period and the number of connected nodes determine the network capacity. The following is a code example:

Sample usage (C#)

```
// Change the sampling interval of node 4228 to 2-second intervals.
This.m_varSampling1.Value = 114;
// Change the sampling period of node 6480 to 256Hz cycle
This.m_varSampling2.Value = 105;
```

4.2.1.4. **Acquires the network capacity after the sampling period is set.**

After setting the sampling period for each node, acquire the network capacity again. The following is a code example:

Sample usage (C#)

```
// Retrieve network capacity
Value = this.m_varNetworkCapacity.Value as float?;
```

4.2.1.5. **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:

Sample usage (C#)

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

4.3. Sample Programming to End Synchronous Sampling of Nodes

This example shows a sample program that starts synchronous sampling and then explicitly terminates synchronous sampling. Table 4-3 describes the requirements of the sample program, and Fig. 4-3 describes the flow of the sample program.

Table 4-3 Sample program requirements

Requirements	Description
Host	Link with serial communications
	The host COM port number is 5.
	The node address numbers are 4228 and 6480.
Process Description	Start synchronous sampling
	End synchronous sampling

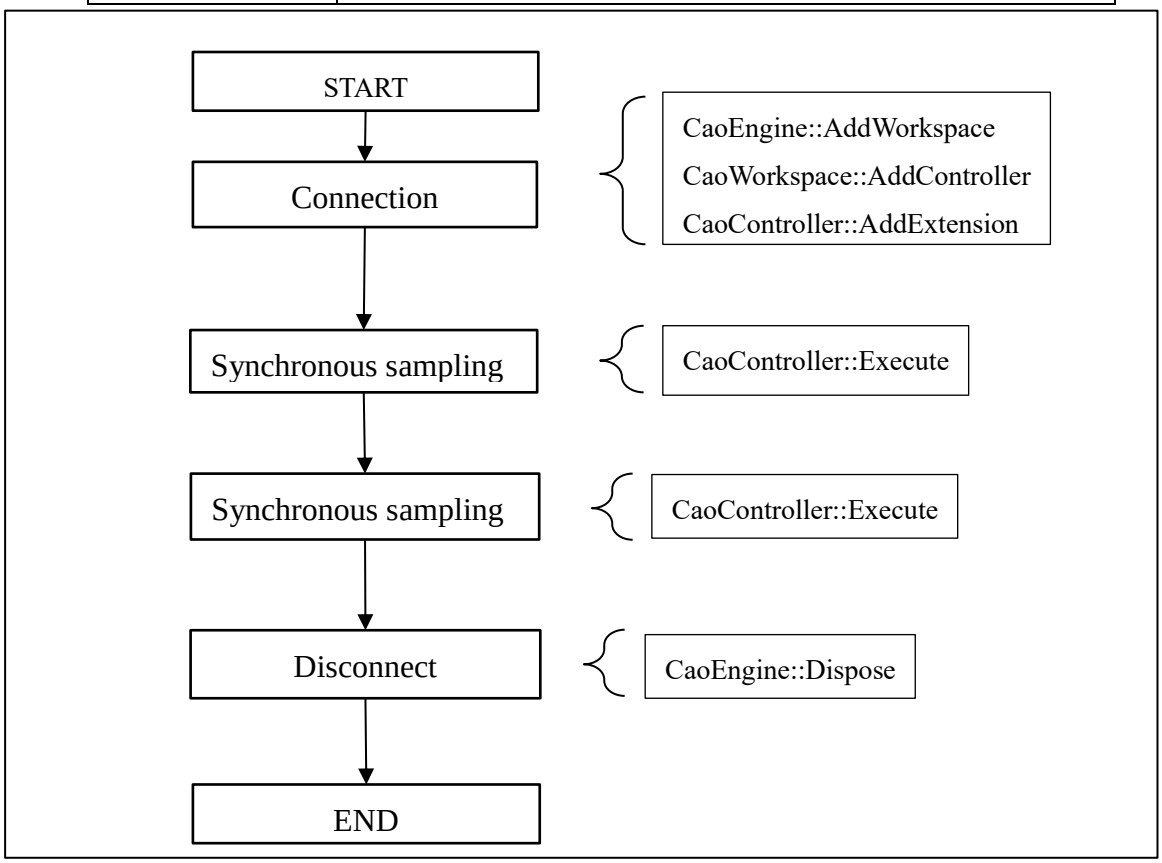


Fig. 4-3 Flow of starting synchronous sampling and acquiring channel data of a node

Specific codes are given in the following sections.

4.3.1. Sample program

The following is an overview of the sample program.

Sample	Sample1.cs

```
// Object
Private ORiN2.ManagedCAO.CCaoEngine m_caoEngine = null;
Private ORiN2.ManagedCAO.CCaoWorkspace m_caoWorkspace = null;
Private ORiN2.ManagedCAO.CCaoController m_caoController = null;
Private ORiN2.ManagedCAO.CCaoExtension m_extNode4228 = null;
Private ORiN2.ManagedCAO.CCaoExtension m_extNode6480 = null;

Public void Main()
{
    // Connection
    This.Connect();

    // Synchronous sampling start
    Object resultStatus = this.m_caoController.Execute("SyncStartSampling", null) as object;

    // Node Status After Synchronous Sampling
    If (resultStatus != null)
    {
        If (resultStatus is ushort[])
        {
            Ushort[] status = resultStatus as ushort[];
            For (int i = 0; i < status.Length; i++)
            {
                // Status of each node (address number ascending order)
                Ushort state = status[i];
            }
        }

    }

    // Synchronous sampling end
    Object nodeStatus = this.m_caoController.Execute("BroadcastSetToIdle", null) as object;
```

```
        // Node Status After Synchronous Sampling
        If (nodeStatus != null)
        {
            If (nodeStatus is ushort[])
            {
                Ushort[] status = nodeStatus as ushort[];
                For (int i = 0; i < status.Length; i++)
                {
                    // Status of each node (address number ascending order)
                    Ushort state = status[i];
                }
            }
        }
        // Disconnect
        This.Disconnect();
    }

    ///<summary>
    /// Connection method
    ///</summary>
    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("WirelessGateway",
                                                                 "CaoProv.LORD.WirelessMSCL",
                                                                 "",
                        "Conn=COM:5,DataCollectionCycle=30,IsBroadcastSetToIdle=true");
        // Creating CaoExtension Objects for Node 4228
        This.m_extNode4228 = this.m_caoController.AddExtension("Node4228", "AddressNo=4228");
        // Creating CaoExtension Objects for Node 6480
        This.m_extNode6480 = this.m_caoController.AddExtension("Node6480", "AddressNo=6480");
    }
```

```
///<summary>
/// Disconnect method
///</summary>
Private void Disconnect()
{
    This.m_caoEngine.Dispose();
    This.m_caoEngine = null;
}
```

4.3.1.1. **Connection**

To connect to the gateway and nodes, proceed as follows:

(1) Prepare a variable to hold the object.

The objects required to connect to the gateway are CaoEngine objects, CaoWorkspace objects, and CaoController objects.CaoWorkspace object does not need to have a variable to retrieve CaoController object from CaoWorkspaces. It also requires a CaoExtension to connect to the node.

Sample usage (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;
// Variables for CaoExtension
Private ORiN2.ManagedCAO.CCaoExtension m_extNode4228 = null;
Private ORiN2.ManagedCAO.CCaoExtension m_extNode6480 = null;
```

(2) Creates a CaoEngine.

CaoEngine is generated using the New keyword.

Sample usage (C#)

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

(3) Gets or generates a CaoWorkspace container.

When you create a CaoEngine object, it defaults to generating one CaoWorkspaces object and one CaoWorkspace object. The following is a sample code/default CaoWorkspace for creating a new

CaoWorkspace.

Sample usage (C#)

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

(4) Creates a CaoController.

To generate a CaoController object, configure the provider name to use and the parameters to use. For WirelessMSCL providers, optionally specify the COM port number of the connected gateway, the frequency at which WirelessMSCL provider periodically collects data to the gateway, and whether nodes in the same frequency band as the gateway are idle when connected. The following is a code example:

Sample usage (C#)

```
// Generate CaoController object
This.m_caoController = this.m_caoWorkspace.AddController("WirelessGateway",
                                                         "CaoProv.LORD.WirelessMSCL",
                                                         "",
                                                         "Conn=COM:5,DataCollectionCycl
e=30,IsBroadcastSetToIdle=true");
```

(5) Creates a CaoExtension.

Create a CaoExtension for the node you want to connect to. The following shows a sample code for generating a Extension object that accesses a node with node address number 4228 and node address number 6480.

Sample usage (C#)

```
// Creating CaoExtension Objects for Node 4228
This.m_extNode4228 = this.m_caoController.AddExtension("Node4228", "AddressNo=4228");
// Creating CaoExtension Objects for Node 6480
This.m_extNode6480 = this.m_caoController.AddExtension("Node6480", "AddressNo=6480");
```

4.3.1.2. **Starting Synchronous Sampling**

Use the connected gateway and nodes to start synchronous sampling. By starting synchronous sampling, data is sent from the connected node to the gateway. The following is a code example:

Sample usage (C#)

```
// Synchronous sampling start
Object resultStatus = this.m_caoController.Execute("SyncStartSampling", null) as object;
// Node Status After Synchronous Sampling
```

```
If (resultStatus != null)
{
    If (resultStatus is ushort[])
    {
        Ushort[] status = resultStatus as ushort[];
        For (int i = 0; i < status.Length; i++)
        {
            // Status of each node (address number ascending order)
            Ushort state = status[i];
        }
    }
}
```

4.3.1.3. **End synchronous sampling**

Explicitly terminates synchronous sampling after starting synchronous sampling. The following is a code example:

Sample usage (C#)

```
// Synchronous sampling end
Object nodeStatus = this.m_caoController.Execute("BroadcastSetToIdle", null) as object;

// Node Status After Synchronous Sampling
If (nodeStatus != null)
{
    If (nodeStatus is ushort[])
    {
        Ushort[] status = nodeStatus as ushort[];
        For (int i = 0; i < status.Length; i++)
        {
            // Status of each node (address number ascending order)
            Ushort state = status[i];
        }
    }
}
```

4.3.1.4. **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:

Sample usage (C#)

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

5. WirelessMSCL Provider Error Codes

This provider has the following proprietary error codes masked with the 0x8011**** (seeTable 5-1 Unique Error Codes). In addition, the mask value of the error code varies depending on where it occurs. The error code that occurs in CaoWorkspace class is 0x80110 The error code that occurs in ***, CaoController class is 0x80111 The error code that occurs in ***, CaoExtension class is 0x80112 The error code that occurs in ***, CaoVariable class is 0x80113***.

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

Table 5-1 Unique Error Codes

Error Number	Description	Countermeasure
0x80110001	Conn optional specification is invalid.	Check 3.2.1.1.1.Conn Optional and specify the correct options.
0x80110002	Timeout optional specification is invalid.	Check 3.2.1.1.AddController method and specify the correct options.
0x80110003	DataCollectionCycle optional specification is invalid.	Check 3.2.1.1.AddController method and specify the correct options.
0x80110004	IsBroadcastSetToIdle optional specification is invalid.	Check 3.2.1.1.AddController method and specify the correct options.
0x80110005	Failed to connect to the gateway.	You may have specified a Com port number to which the gateway is not connected. Specify the correct Com port number.
0x80110006	An attempt to connect to an already connected gateway failed.	Specify a gateway that does not exist in Controller.
0x80111001	Could not start synchronous sampling.	Check the device status and node status.
0x80111002	Network usage in synchronous sampling exceeded.	Re-set the number of nodes and sampling period of nodes so that they are 100% or less.
0x80111003	A command name that does not exist was specified.	Check 3.3.1.CaoController class-command and specify the correct options.
0x80111011	A variable name that does not exist was specified.	Check 3.4.2.CaoController class-variable and specify the correct options.
0x80111021	The address number is unspecified.	Check 3.2.2.3.AddExtension method and specify the correct options.
0x80111022	The address number value is out of range.	Check 3.2.2.3.AddExtension method and specify the correct options.

Error Number	Description	Countermeasure
0x80111023	An existing address number was specified.	Specify an address number other than the node address number of Extension class that exists in Controller class.
0x80111024	Sampling period value is out of range.	Check 3.2.2.3.AddExtension method and specify the correct options.
0x80112001	A command name that does not exist was specified.	Check 3.3.2.CaoExtension class-command and specify the correct options.
0x80112011	A variable name that does not exist was specified.	Check 3.4.3.CaoExtension class-variable and specify the correct options.
0x80112012	The channel number is unspecified.	Check 3.4.3.10.COLLECTDATA<??> and specify the correct options.
0x80112013	The channel number value is out of range.	Specify a channel number within the range.
0x80112014	A channel number that already exists was specified.	Specify a channel number other than the channel number of COLLECTDATA <?> variable that exists in Extension class.
0x80113001	Sampling period value is out of range.	Check Table 3-2 List of sampling intervals for the specifiable nodes and specify the correct options.

In addition, this provider returns the error code of the error that occurred in the API as it is.
For more information about API failures, see MSCL documentation.

Appendix A. API Compatibility Table

CaoWorkspace::AddController

API function name
Mscl.Connection.Serial
Mscl.Devices.listBaseStations
Mscl.BaseStation.ping
Mscl.BaseStation.broadcastSetToIdle ※ When a IsBroadcastSetToIdle=true is specified as an optional CaoWorkspace::AddController

CaoController (during periodic sampling)

API function name
Mscl.BaseStation.getData
Mscl.DataSweep.nodeAddress
Mscl.DataSweep.timestamp
Mscl.DataSweep.data
Mscl.WirelessDataPoint.channelId
Mscl.WirelessDataPoint.storedAs
Mscl.WirelessDataPoint.as_bool ※ If the datatype is bool
Mscl.WirelessDataPoint.as_Bytes ※ If the datatype is Bytes
Mscl.WirelessDataPoint.as_ChannelMask Mscl.ChannelMask.enabled ※ If the datatype is ChannelMask
Mscl.WirelessDataPoint.as_double ※ If the datatype is double
Mscl.WirelessDataPoint.as_float ※ If the datatype is float
Mscl.WirelessDataPoint.as_int16 ※ If the datatype is int16
Mscl.WirelessDataPoint.as_int32 ※ If the datatype is int32
Mscl.WirelessDataPoint.as_int8 ※ If the datatype is int8
Mscl.WirelessDataPoint.as_Matrix

API function name
Mscl.Matrix.valuesType Mscl.Matrix.rows Mscl.Matrix.columns Mscl.Matrix if the datatype inside Matrix is double Mscl.Matrix if the datatype inside Matrix is float Mscl.Matrix if the datatype inside Matrix is uint16 Mscl.Matrix if the datatype inside Matrix is uint8 ※ If the datatype is Matrix
Mscl.WirelessDataPoint.as_RfSweep ※ If the datatype is RfSweep
Mscl.WirelessDataPoint.as_string ※ If the datatype is string
Mscl.WirelessDataPoint.as_StructuralHealth Mscl.StructuralHealth.angle Mscl.StructuralHealth.uptime Mscl.StructuralHealth.proccessingRate Mscl.SampleRate.rateType Mscl.SampleRate.samples Mscl.StructuralHealth.histogram Mscl.Histogram.bins Mscl.Bin.start Mscl.Bin.end Mscl.Bin.count ※ If the datatype is StructuralHealth
Mscl.WirelessDataPoint.as_Timestamp ※ If the datatype is Timestamp
Mscl.WirelessDataPoint.as_uint16 ※ If the datatype is uint16
Mscl.WirelessDataPoint.as_uint32 ※ If the datatype is uint32
Mscl.WirelessDataPoint.as_uint8 ※ If the datatype is uint8
Mscl.WirelessDataPoint.as_Vector Mscl.Vector if the datatype inside Matrix is double Mscl.Vector if the datatype inside Matrix is float

API function name
Mscl.Vector if the datatype inside Matrix is uint16
Mscl.Vector if the datatype inside Matrix is uint8
※ If the datatype is Vector

CaoController::Execute

Command name	API function name
BroadcastSetToIdle	Mscl.BaseStation.broadcastSetToIdle
	Mscl.WirelessNode.lastDeviceState
SyncStartSampling	Mscl.SyncSamplingNetwork.removeNode
	Mscl.SyncSamplingNetwork.addNode
	Mscl.SyncSamplingNetwork.percentBandwidth
	Mscl.SyncSamplingNetwork.ok
	Mscl.SyncSamplingNetwork.applyConfiguration
	Mscl.SyncSamplingNetwork.startSampling
	Mscl.WirelessNode.lastDeviceState

CaoController::AddExtension

API function name
Mscl.WirelessNodeConfig.dataCollectionMethod
Mscl.WirelessNodeConfig.sampling
Mscl.WirelessNodeConfig.sampleRate
※ When CaoController::AddExtension optional Sampling is specified
Mscl.WirelessNode.applyConfig

CaoController::Execute

Command name	API function name
SetToIdle	Mscl.WirelessNode.setToIdle
	Mscl.WirelessNode.lastDeviceState
Ping	Mscl.WirelessNode.ping
GetActiveChannels	Mscl.WirelessNode.getActiveChannels

CaoVariable

Parent class	Variable Name	API function name
CaoController	@NETWORKCAPACITY	Mscl.SyncSamplingNetwork.percentBandwidth
CaoController	@COMMPROTOCOL	Mscl.BaseStation.communicationProtocol

Parent class	Variable Name	API function name
CaoController	@SERIALNO	Mscl.BaseStation.serial
CaoController	@CLOUD_NAME	Mscl.BaseStation.name
CaoController	@FIRMWARE_VERSION	Mscl.BaseStation.firmwareVersion
CaoController	@MODEL	Mscl.BaseStation.model
CaoExtension	@COMMPROTOCOL	Mscl.WirelessNode.communicationProtocol
CaoExtension	@SERIALNO	Mscl.WirelessNode.serial
CaoExtension	@CLOUD_NAME	Mscl.WirelessNode.name
CaoExtension	@FIRMWARE_VERSION	Mscl.WirelessNode.firmwareVersion
CaoExtension	@MODEL	Mscl.WirelessNode.model
CaoExtension	@SAMPLING	When get_value: Mscl.WirelessNode.getSampleRate For put_value: Mscl.WirelessNodeConfig.sampleRate Mscl.WirelessNode.applyConfig
CaoExtension	@LASTSTATUS	Mscl.WirelessNode.lastDeviceState

Appendix B. Points to note and remedy when connecting to nodes

When connected to the gateway, if the node is in the sampling state or sleep state, it is not possible to connect to the node (synchronous sampling, node information, etc. cannot be acquired).

The following table describes how to handle each problem.

The node is in the sampling state when connected to the gateway.

3.2.1.1. Specifying the optional IsBroadcastSetToIdle at the time of AddController execution in true or executing the 3.3.1.1. BroadcastSetToIdle command enables connection from the sampled state to the idle state.

The node is in sleep state when connected to the gateway.

If the node is in sleep state when connected to the gateway, the operation cannot be performed from the client PC, etc. Therefore, you can connect from the node's switch by physically going from sleep to idle.

Appendix C. Approximate times before an OnMessage occurs

If accumulated data is not fetched after synchronous sampling is started, the time at which a 3.5.1.1. data memory capacity overmessage occurs depends on the number of connected gateways, the network usage, and the sampling period. Conditions and time estimates are shown below.

- **Common environment**
  Gateway: 1
  Node: 1

1. When the communication protocol is LXRS

Sampling period of the node	Network usage in the frequency band	How long OnMessage will occur from the beginning of the sample
512Hz	100%	About 34 minutes
256Hz	50%	Approximately 68 minutes
128Hz	25%	About 136 minutes

2. When the communication protocol is LXRS +.

Sampling period of the node	Network usage in the frequency band	How long OnMessage will occur from the beginning of the sample
2048Hz	100%	About 8 minutes and 30 seconds
1024Hz	50%	About 17 minutes
512Hz	25%	About 34 minutes