Parker 社
WirelessMSCL プロバイダ
ユーザーズ ガイド

Version 1.0.1

July 14, 2022

備考:

© 2021 DENSO WAVE INCORPORATED
この取扱説明書の著作権は，株式会社デンソーウェーブにあります．
本書に掲載されている会社名や製品は，一般に各社の商標または登録商標です．

この取扱説明書の一部または全部を無断で複製・転載することはお断りします．
- この説明書の内容は将来予告なしに変更することがあります．
- 本書の内容については，万全を期して作成いたしましたが，万一ご不審の点や誤り，記載もれなど，お気づきの点がありましたらご連絡ください．
- 運用した結果の影響については，上項にかかわらず責任を負いかねますのでご了承ください．

【改版履歴】

バージョン	日付	内容
1.0.0	2021-08-27	初版.
1.0.1	2022-07-14	ドキュメント内の LORD の社名を Parker に変更 @MAKER_NAME 変数の取得値を Parker に変更

【対応機種】

機種	バージョン	注意事項
WSDA-2000	---	USB 接続のみ対応
WSDA-Base-200 (USB)	---	---
WSDA-Base-101 (Analog)	---	---
WSDA-Base-104 (USB)	---	---

【動作確認機種】

機種	バージョン	注意事項
WSDA-Base-200 (USB)	6.44079	

目次

1. はじめに

本書は，Parker 社のワイヤレスノード（以降，ノードと呼称）から，チャネルなどのデータを収集するワイヤレスゲートウェイ（以降，ゲートウェイと呼称）に対して，データを収集するプロバイダのユーザーズガイドです．図 1-1 が本プロバイダとデバイスの全体構成図になります．本プロバイダは，Parker 社製の MicroStrain 通信ライブラリである MicroStrain Communication Library(MSCL)を使用して，ゲートウェイと接続等を行います．MSCL は，ゲートウェイと通信するために，シリアル通信（USB 接続）を使用します．以降本プロバイダを WirelessMSCL プロバイダと呼称します．

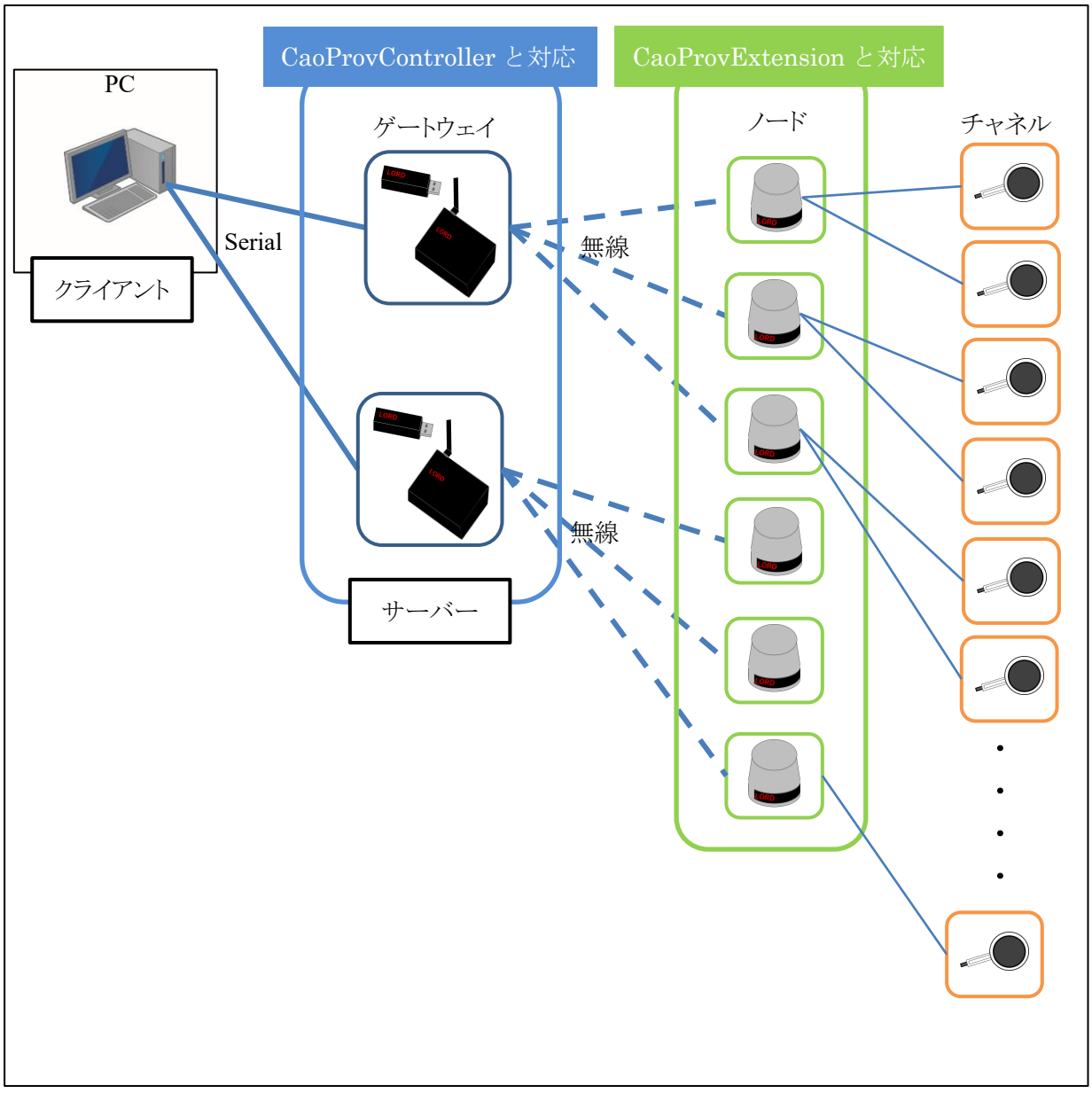


図 1-1 構成図

また，本プロバイダ及びデバイスそれぞれの対応を図 1-2 に表します．

(※一例です. 全てを表しているわけではありません.)

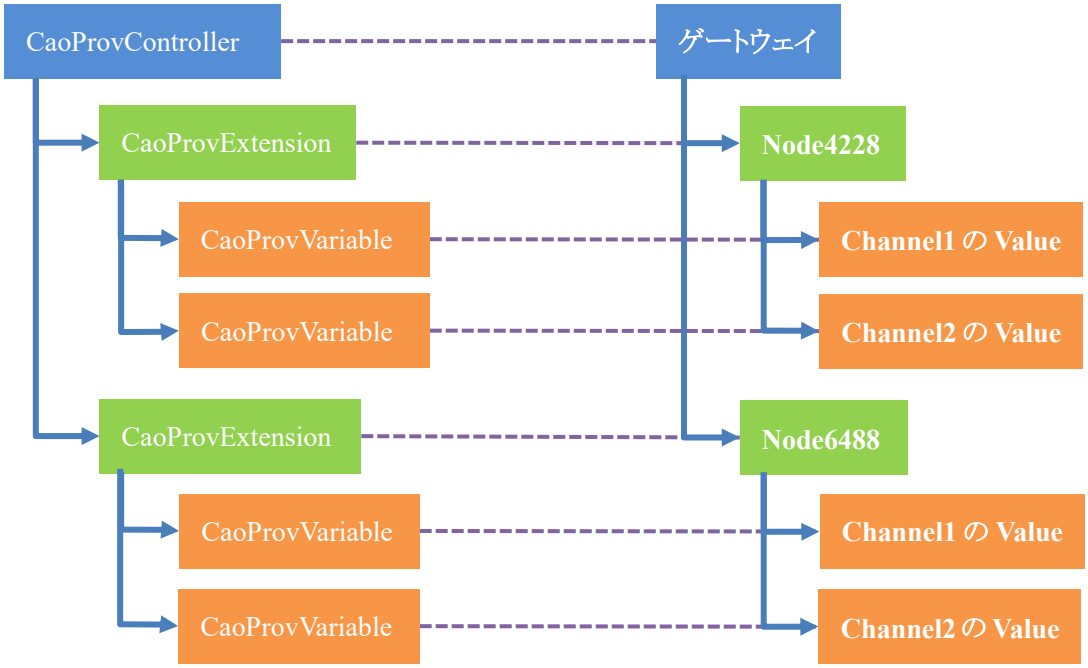


図 1-2 プロバイダの構成とデバイス情報との対応図

1.1. 使用するライブラリについて

WirelessMSCL プロバイダでは, ワイヤレスゲートウェイとの通信に使用するために Parker 社製の OSS である MSCL を使用しています.

MSCL に関しては, 以下のサイトを参照してください.

[関連サイトリンク]

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

[MSCL の著作権とライセンス]

Copyright(c) 2022 Parker Hannifin Corp.

All rights reserved.

MIT License

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

2. アプリケーション開発のための環境セットアップ

2.1. ゲートウェイとクライアント PC との接続準備

2.1.1. クライアント PC ゲートウェイとの接続

ゲートウェイとクライアント PC はシリアル通信にて接続を行います．また接続には通信ドライバが必要となります．通信ドライバのインストールについては，下記サイトを参照してください．

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

2.1.2. SensorConnect アプリケーションのインストール

ゲートウェイとノードを設定する際に Parker 社製の設定アプリケーション「SensorConnect」が必要となります．「SensorConnect」を，下記 URL の Parker 社のホームページからダウンロードし，インストールしてください．アプリケーションのインストールと同時に通信ドライバもインストールされます．

https://www.microstrain.com/software

2.1.3. ゲートウェイとノードの通信プロトコル，周波数帯の設定

Parker 社製の設定アプリケーション「SensorConnect」を利用して，接続するゲートウェイ，ノードの通信プロトコル，周波数帯を環境に合わせて設定します．下図は周波数帯の概念図になります．

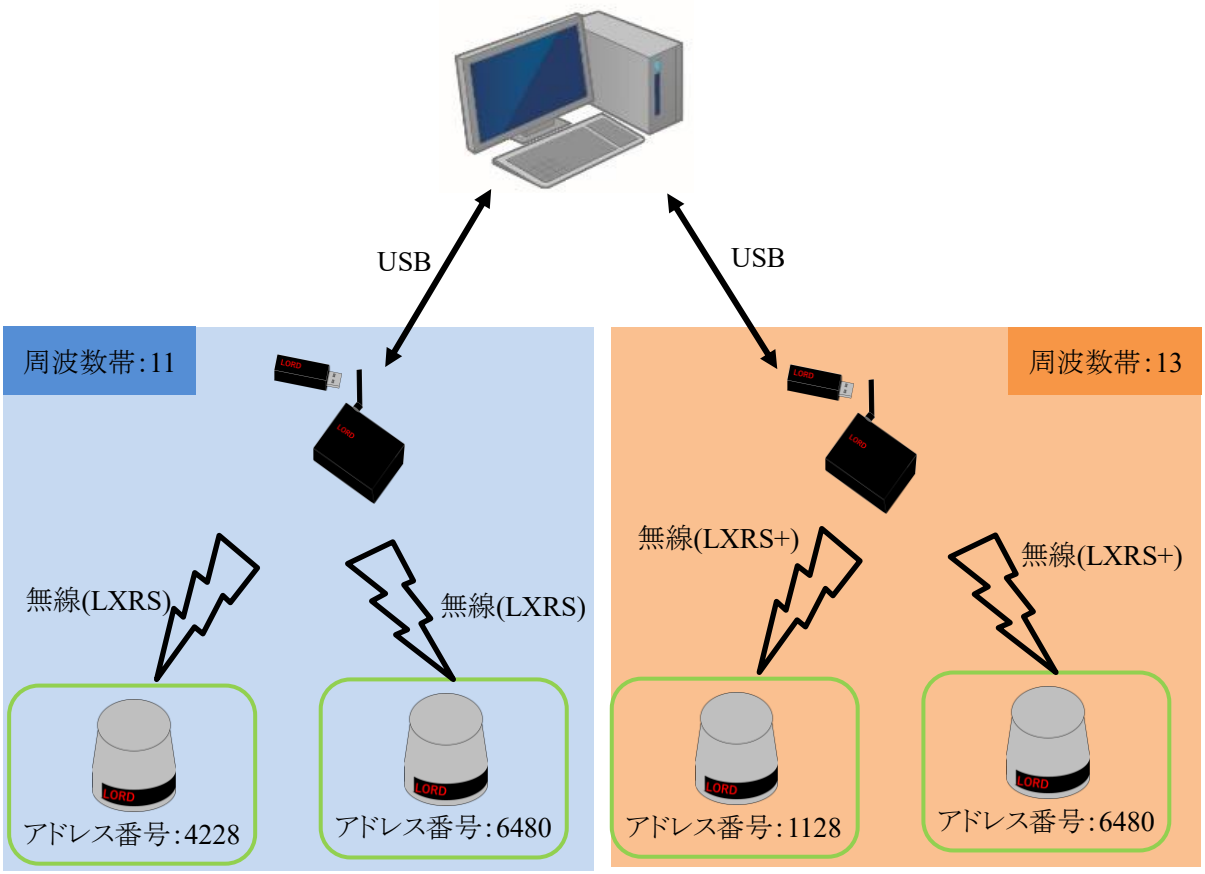


図 2-1 通信プロトコルと周波数帯の関係

通信プロトコル

ゲートウェイとノードの間の無線通信の方式です．通信方式は LXRS と LXRS+が存在します．

・LXRS

LXRS はネットワークのスループットよりも通信距離を優先し，最大 2km の通信距離と最大 4k サンプル/秒のネットワークスループットで動作するように設計されています．

・LXRS＋

LXRS+は通信距離よりもネットワークのスループットを優先し，最大 400m の距離で最大 16k サンプル/秒のネットワークスループットで動作するように設計されています．

周波数帯

ゲートウェイとノードの間の無線通信に割り当てる周波数帯です．同じ周波数帯のゲートウェイとノードが通信可能です．

周波数帯	周波数(バンド幅)
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. PC 開発環境のセットアップ

2.2.1. MSCL の準備

WirelessMSCL プロバイダは, Parker 社から提供されている MSCL を使用します. MSCL は ORiN2 をインストールした場合, 自動でインストールされます.

表 2-1 使用する MSCL

DLL
MSCL.dll
MSCL_Managed.dll

2.2.2. WirelessMSCL プロバイダの手動インストール

WirelessMSCLプロバイダを手動でインストールする場合は下記レジストリ登録を行う必要があります. そのファイル形式はEXEであり, CAOエンジンから使用時に動的にロードされます. WirelessMSCLプロバイダを使用するには表2-2の方法で登録を行う必要があります. RegistAsm.batおよびUnregistAsm.batはORiN2SDKをインストールしたフォルダの下のDotNet¥BATフォルダにあります.

表 2-2 WirelessMSCL プロバイダ

ファイル名	CaoProvLORDWirelessMSCL.exe
ProgID	CaoProv.LORD.WirelessMSCL
レジストリ登録	RegistAsm.bat CaoProvLORDWirelessMSCL.exe
レジストリ登録の抹消	UnregistAsm.bat CaoProvLORDWirelessMSCL.exe

3. コマンドリファレンス

3.1. メソッド/プロパティー覧

表 3-1 メソッド/プロパティー覧

カテゴリ	メソッド/プロパティ[1]		機能	参照
CaoWorkspace				
	AddController	M	コントローラに接続	P.13
CaoController				
	GetVariableNames	M	接続可能な変数名リストの取得	P.16
	Variables	P	コントローラが保持する変数コレクションの取得	P.16
	AddExtension	M	拡張ボードオブジェクトの追加	P.16
	AddVariable	M	変数オブジェクトの追加	P.19
	Execute	M	拡張コマンドの実行	P.19
	OnMessage	E	メッセージ受信イベント	P.20
CaoExtenstion				
	GetVariableNames	M	接続可能な変数名リストの取得	P.20
	Variables	P	タスクが保持する変数コレクションの取得	P.20
	AddVariable	M	変数オブジェクトの追加	P.21
	Execute	M	拡張コマンドの実行	P.21
CaoVariable				
	Value	P	値の取得/設定	P.21

3.2. メソッド・プロパティ

3.2.1. CaoWorkspace クラス

3.2.1.1. AddController メソッド

CaoWorkspace に，コントローラオブジェクトを追加します．WirelessMSCL プロバイダでは，AddController メソッド実行時に渡されたパラメータを参照し，該当するゲートウェイと接続を行います．以下に，AddController メソッドの仕様を示します．

書式

```
AddController
(
"<コントローラ名>",                    // コントローラ名(任意)
```

[1] M:メソッド, P:プロパティ, E:イベントをそれぞれ示します．

```
"CaoProv.LORD.WirelessMSCL",        // プロバイダ名（固定）
"<マシン名>",                        // プロバイダ実行マシン名(未使用)
"<オプション>"                       // オプション文字列
)
```

オプション

以下にオプション文字列に指定するオプションを示します．オプション文字列は下記に示す各オプションをカンマ(,)でつなげた文字列となります．

オプション	必須	説明	値範囲	デフォルト値
Conn	○	接続先情報を指定します．	-----	-----
Timeout	-	タイムアウト時間(ms)を指定します．	1 - 65535	300
DataCollectionCycle	-	ゲートウェイからデータを収集する周期(ms)を指定します．詳細は 3.2.1.1.2 を参照してください．	1 - 30000	10
IsBroadcastSetToIdle	-	接続時にブロードキャストにてノードをアイドル状態にするコマンドを実行します．詳細は 3.2.1.1.3 を参照してください．	true,false	true

使用例(C#)

```
// Engine オブジェクト
ORiN2.ManagedCAO.CCaoEngine engine = new ORiN2.ManagedCAO.CCaoEngine();
// Workspace オブジェクト
ORiN2.ManagedCAO.CCaoWorkspace workspace = engine.AddWorkspace("NewWrks", "");
// Controller オブジェクト
ORiN2.ManagedCAO.CCaoController controller= workspace.AddController(Basestation1,
                                    "CaoProv.LORD.WirelessMSCL",
                                    "",
                                    "Conn = COM:1,Timeout = 200, DataCollectionCycle = 20");
```

3.2.1.1.1. Conn オプション

以下に Conn オプションの接続パラメータ文字列を示します．

・Serial 通信

"Conn=COM:<COM Port>"

<COM Port> : COM ポート番号． '1' - COM1, '2' - COM2, …

3.2.1.1.2. 定期的にゲートウェイからデータを収集

WirelessMSCL プロバイダでは図 3-1 のように，AddController 後にゲートウェイから定期的にデータを収集します．3.3.1.2 にて同期サンプリングを開始すると，同期サンプリング内のノードが取得したデータをゲートウェイに送信します．ゲートウェイは送信されたデータを一度保持します．本プロバイダは保持されたゲートウェイのデータを AddController 時に指定した DataCollectionCycle オプションの周期で定期的にゲートウェイからデータを収集し，定期サンプリングを行います．

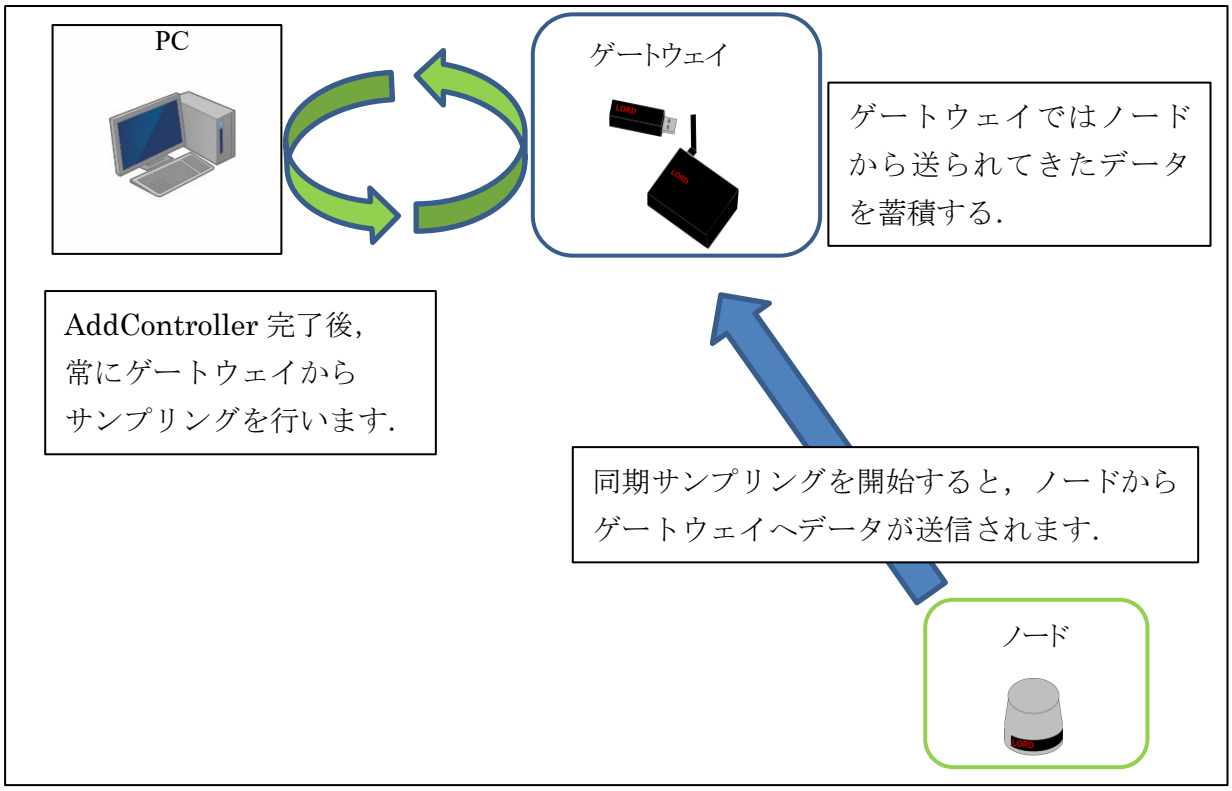


図 3-1 定期サンプリングイメージ図

定期サンプリングにて収集したデータは，3.2.3.CaoExtension クラスに対応したノードの診断情報であれば最新の診断情報を保持し，チャネルデータの場合，同じチャネル番号の 3.4.3.10.COLLECTDATA<??>変数に対応した蓄積データとして保持します．この蓄積されたチャネルデータは，3.4.3.10.COLLECTDATA<??>変数の get_value が実行されるまで蓄積し続け，get_value が実行されると蓄積しているデータを取り出し，蓄積データを削除します．

本プロバイダの使用方法については，4.WirelessMSCL プロバイダによるプログラミングを参照してください．

3.2.1.1.3. 接続時の注意点

本プロバイダでは，AddController 時に IsBroadcastSetToIdle オプションを省略もしくは true を指定した場合，ゲートウェイと接続後ゲートウェイにブロードキャストにてノードをアイドル状態にするコマンドを実行させます．このコマンドを実行することによって，予期しない形でゲートウェイと切断した場合に再接続時にサンプリング状態のままのノードをアイドル状態へ遷移させることができます．

3.2.1.1.4. ゲートウェイとクライアント PC で切断が起きた場合

ゲートウェイとクライアント PC 間の接続が何らかの理由で切断された場合，再度ゲートウェイとクライアント PC が接続されても，同じ Controller クラスでは値の取得などはできません．一度対象の Controller クラスのオブジェクトを削除し，Controller クラスを作り直してください．

3.2.2. CaoController クラス

3.2.2.1. GetVariableNames メソッド

接続可能な変数名リストを取得します．

書式

```
GetVariableNames
(
    "<オプション>"                    // オプション文字列(未指定)
)
```

使用例(C#)

```
// 変数名リスト取得
string[] variableNames = controller.GetVariableNames("");
```

3.2.2.2. Variables プロパティ

コントローラが保持する，変数コレクションを取得します．

使用例(C#)

```
// 変数コレクション取得
ORiN2.ManagedCAO.CCaoVariables variables = controller.Variables;
// 変数取得
ORiN2.ManagedCAO.CCaoVariable variable = variables[0];
```

3.2.2.3. AddExtension メソッド

CaoController に，ノードに対応した拡張ボードオブジェクトを追加します．オブジェクト追加時はノードとの接続に失敗しても追加に成功します．ノードとの接続は，CaoExtension クラスの Execute メソッドの実行，または CaoExtension クラス変数の @COMMPROTOCOL，@SERIALNO，@CLOUD_NAME，@FIRMWARE_VERSION，@MODEL，@SAMPLING の取得/設定を行うことで接続されます．またノードの状態は@LASTSTATUS にて確認できます．

ノードとの接続時の注意点を 5.付録 B.ノードとの接続時の注意点と対処法に記載しておりますので参照してください．

以下に，AddExtension メソッドの仕様を示します．

書式

```
AddExtension
(
"<拡張ボード名>",          // 拡張ボード名(任意)
"<オプション>"             // オプション文字列
)
```

オプション

以下にオプション文字列に指定するオプションを示します. オプション文字列は下記に示す各オプションをカンマ(,)でつなげた文字列となります.

オプション	必須	説明	値範囲	デフォルト値
AddressNo	○	対象のノードのアドレス番号を指定します. 同一ゲートウェイに同じアドレス番号のノードは 1 つのみ追加可能です.	1 -	-----
Sampling	-	ノードのサンプリング周期を指定します. 指定可能な値は表 3-2 から指定してください.	表 3-2 を参照	機器に設定されているサンプリング周期を使用します.

※ サンプリング周期はアイドル状態のノードにのみ設定可能です.

使用例(C#)

```
// 拡張ボードの追加
ORiN2.ManagedCAO.CCaoExtension caoExt = controller.AddExtension("Node4228",
                                                "AddressNo=4228,Sampling=47");
```

表 3-2 指定可能なノードのサンプリング周期一覧

値	意味
46	300Hz
47	800Hz
48	1,600Hz
49	3,200Hz
55	12,500Hz
56	25,000Hz
57	62,500Hz

値	意味
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
100	8,192Hz
101	4,096Hz
102	2,048Hz
103	1,024Hz
104	512Hz
105	256Hz
106	128Hz
107	64Hz

値	意味
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 メソッド

CaoController に変数オブジェクトを追加します. 変数名には 3.4.2 に示すもののみ使用できます.
以下に, AddVariable の仕様を示します.

書式

```
AddVariable
(
"<変数名>",                    // 変数名
"<オプション>"                 // オプション文字列(省略可能)
)
```

3.2.2.5. Execute メソッド

CaoController クラスの拡張コマンドを実行します. コマンド名には 0 に示すもののみ使用できます.
以下に, Execute の仕様を示します.

書式

```
Execute
(
"<拡張コマンド名>",          // 拡張コマンド名
"<オプション文字列>"         // オプション文字列(省略可能)
)
```

3.2.2.6. OnMessage イベント

コントローラのエラー通知や状態の変化を OnMessage イベントとして受け取ることが可能です. 受け取れるイベントについては 3.5 を参照してください.

3.2.3. CaoExtension クラス

ノードに対応した拡張ボードオブジェクトです. 本クラスのオブジェクトを削除する際に, 対象ノードに対して SetToIdle コマンドを実行し, サンプリング状態等の状態からアイドル状態にします.

3.2.3.1. GetVariableNames メソッド

接続可能な変数名リストを取得します.

書式

```
GetVariableNames
(
"<オプション>"                    // オプション文字列(未使用)
)
```

使用例(C#)

```
// 変数名リスト取得
string[] variableNames = controller.GetVariableNames("");
```

3.2.3.2. Variables プロパティ

拡張ボードが保持する, 変数コレクションを取得します.

使用例(C#)

```
// 変数コレクション取得
ORiN2.ManagedCAO.CCaoVariables variables = caoExt.Variables;
// 変数取得
ORiN2.ManagedCAO.CCaoVariable variable = variables[0];
```

3.2.3.3. AddVariable メソッド

CaoExtension に変数オブジェクトを追加します. 変数名には 3.4.3 に示すもののみ使用できます.
以下に, AddVariable の仕様を示します.

書式

```
AddVariable
(
"<変数名>",          // 変数名
"<オプション>"       // オプション文字列(省略可能)
)
```

3.2.3.4. Execute メソッド

CaoExtension クラスの拡張コマンドを実行します. コマンド名には 3.3.2 に示すもののみ使用できます.
以下に, Execute の仕様を示します.

書式

```
Execute
(
"<拡張コマンド名>",          // 拡張コマンド名
"<オプション文字列>"         // オプション文字列(省略可能)
)
```

3.2.4. CaoVariable クラス

3.2.4.1. Value プロパティ

接続したゲートウェイまたはノードからデータを取得/設定します. 変数名によって動作が異なります. 詳細は, 3.4.変数一覧を参照してください.

3.3. 拡張コマンド一覧

各クラスで使用可能な拡張コマンドを定義します.

3.3.1. CaoController クラスコマンド

以下に, CaoController クラスの Execute で指定できる拡張コマンド一覧を示します. 使用例は拡張コマンドの詳細で記述しています.

コマンド	説明	参照
BroadcastSettoIdle	ブロードキャストにてノードをアイドル状態にします.	P.22
SyncStartSampling	同期サンプリングを開始します.	P.23

3.3.1.1. BroadcastSetToIdle コマンド

ブロードキャストにてゲートウェイと同周波数帯のすべてノードをアイドル状態にします. また本コマンドは, AddController 時に指定したタイムアウト時間分, 存在する Extension クラスのノードがアイドル状態になるまでステータスを取得し続け, ステータスを返却します. 存在する Extension クラスのノードが, すべてアイドル状態になれば成功です. ノードがアイドル状態でなくても, タイムアウト時間経過後, ステータスを返却します.

実行直後, 物理的なノード状態は変更されますが戻り値のステータスが変更されない場合があります. その場合, 少し時間を置くとステータスに反映されることがあります. @LASTSTATUS の値を取得して再度ノードの状態を確認してください.

以下に引数と戻り値を示します.

データ型

項目	型説明	
引数	なし	
戻り値	VT_ARRAY | VT_UI1	
	i	存在する Extension クラスのノードアドレス番号の昇順にステータスが格納されます. ステータスの詳細は 3.4.3.8 のデータ型を参照してください

※i : CaoContoroller クラスに追加した CaoExtension クラス数

※CaoExtension クラスが存在しない場合は VT_EMPTY が戻り値となります.

使用例(C#)

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

// コマンド実行結果
if (nodeStatus != null)
{
    if (nodeStatus is ushort[])
```

```
        {
            ushort[] status = nodeStatus as ushort[];
            for (int i = 0; i < status.Length; i++)
            {
                // 各ノードのステータス(アドレス番号昇順)
                ushort state = status[i];
            }
        }
    }
```

3.3.1.2. SyncStartSampling コマンド

自身が保持する Extension クラスのノード情報を使用し，ゲートウェイの同期サンプリングを開始させます．また本コマンドは，AddController 時に指定したタイムアウト時間分，存在する Extension クラスのノードがアイドル状態になるまでステータスを取得し続け，ステータスを返却します．存在する Extension クラスのノードが，すべてアイドル状態になれば成功です．ノードがアイドル状態でなくても，タイムアウト時間経過後，ステータスを返却します．

同期サンプリングを終了させるには，ノードをサンプリング状態から，アイドル状態にする必要があります．ノードをアイドル状態にするには，Controller クラスの BroadcastSetToIdle コマンドまたは Extension クラスの SetToIdle コマンドを実行してください．

以下に引数と戻り値を示します．

データ型

項目	型説明	
引数	なし	
戻り値	VT_ARRAY | VT_UI1	
	i	存在する Extension クラスのノードアドレス番号の昇順にステータスが格納されます． ステータスの詳細は 3.4.3.8 のデータ型を参照してください

※i : CaoController クラスに追加した CaoExtension クラス数

※CaoExtension クラスが存在しない場合は VT_EMPTY が戻り値となります．

使用例(C#)

```
// SyncStartSampling コマンドを実行
object resultStatus = controller.Execute("SyncStartSampling", null) as object;
// コマンド実行後のノードステータス
if (resultStatus != null)
{
    if (resultStatus is ushort[])
```

```
        {
            ushort[] status = resultStatus as ushort[];
            for (int i = 0; i < status.Length; i++)
            {
                // 各ノードのステータス(アドレス番号昇順)
                ushort state = status[i];
            }
        }
    }
```

3.3.2. CaoExtension クラスコマンド

以下に，CaoExtension クラスの Execute で指定できる拡張コマンド一覧を示します．使用例は拡張コマンドの詳細で記述しています．

コマンド	説明	参照
SettoIdle	ノードをアイドル状態にします．	P.24
Ping	ノードとの接続確認を行います．	P.25
GetActiveChannels	現在のノードの有効なチャネルを取得します．	P.25
EntrySampling	同期サンプリングのグループに参加します．	P.26

3.3.2.1. SetToIdle コマンド

ノードをサンプリング状態またはスリープ状態からアイドル状態にします．AddController 時に指定したタイムアウト時間分アイドル状態になるまでノードのステータスを確認します．

AddExtension 時にアイドル状態でないノードの場合，本コマンドではノードをアイドル状態にすることはできません．AddExtension 時には必ず，ノードをアイドル状態にしておいてください．AddExtension 時にアイドル状態であれば，本コマンドにてノードをアイドル状態に変更可能です．

以下に引数と戻り値を示します．

データ型

項目	型説明	
引数	なし	
戻り値	VT_UI1	コマンド実行後のノードのステータス．ステータスの詳細は 3.4.3.8 を参照してください．

使用例(C#)

```
// SetToIdle コマンド実行
ushort? status = extension.Execute("SetToIdle", "") as ushort?;
```

3.3.2.2. Ping コマンド

ノードが通信可能か否かを確認します. ノードがアイドル状態の場合のみ成功します. 以下に戻り値を示します.

データ型

項目	型説明	
引数	なし	
戻り値	VT_BOOL	通信成否 true ： 通信可能 false : 通信不可

使用例(C#)

```
// Ping コマンド実行
bool? result = extension.Execute("Ping", "") as bool?;
```

3.3.2.3. GetActiveChannels コマンド

ノードの有効なチャネル番号の一覧を取得します. 取得したチャネル番号は 3.4.3.10.COLLECTDATA<??>変数のオプションにて指定することで対象のチャネルデータを取得することができます.

以下に引数と戻り値を示します.

データ型

項目	型説明	
引数	なし	
戻り値	VT_ARRAY | VT_UI1	
	i	有効なチャネル番号

※ i : 有効なチャネル数分

※ 有効なチャネルが存在しない場合は VT_EMPTY が戻り値となります.

使用例(C#)

```
// GetActiveChannels コマンド実行
object activeChannels = extension.Execute("GetActiveChannels", "") as object;
// 有効なチャネルを確認
if (activeChannels != null)
{
    if (activeChannels is byte[])
    {
```

```
            byte[] channels = activeChannels as byte[];
            for (int i = 0; i < channels.Length; i++)
            {
                // 有効なチャネル番号
                byte activeChannel = channels[i];
            }
        }
    }
```

3.3.2.4. EntrySampling コマンド

ノードを同期サンプリングのグループに参加させます. 同期サンプリングのグループに参加させると同期サンプリング実行時に対象のノードのサンプリングを開始させることができます. AddExtension メソッド実行時に, ノードは同期サンプリングのグループに参加します. ただしノードとの接続に失敗した場合, 同期サンプリングのグループは未参加状態になります. また, 同期サンプリングを実行後, ノードをアイドル状態にした場合は, 再度本コマンドを実行して下さい. 同期サンプリングのグループについては 3.3.2.4.1 を参照してください.

以下に引数と戻り値を示します.

データ型

項目	型説明	
引数	なし	
戻り値	VT_BOOL	参加状態 true ： 参加済み false： 未参加

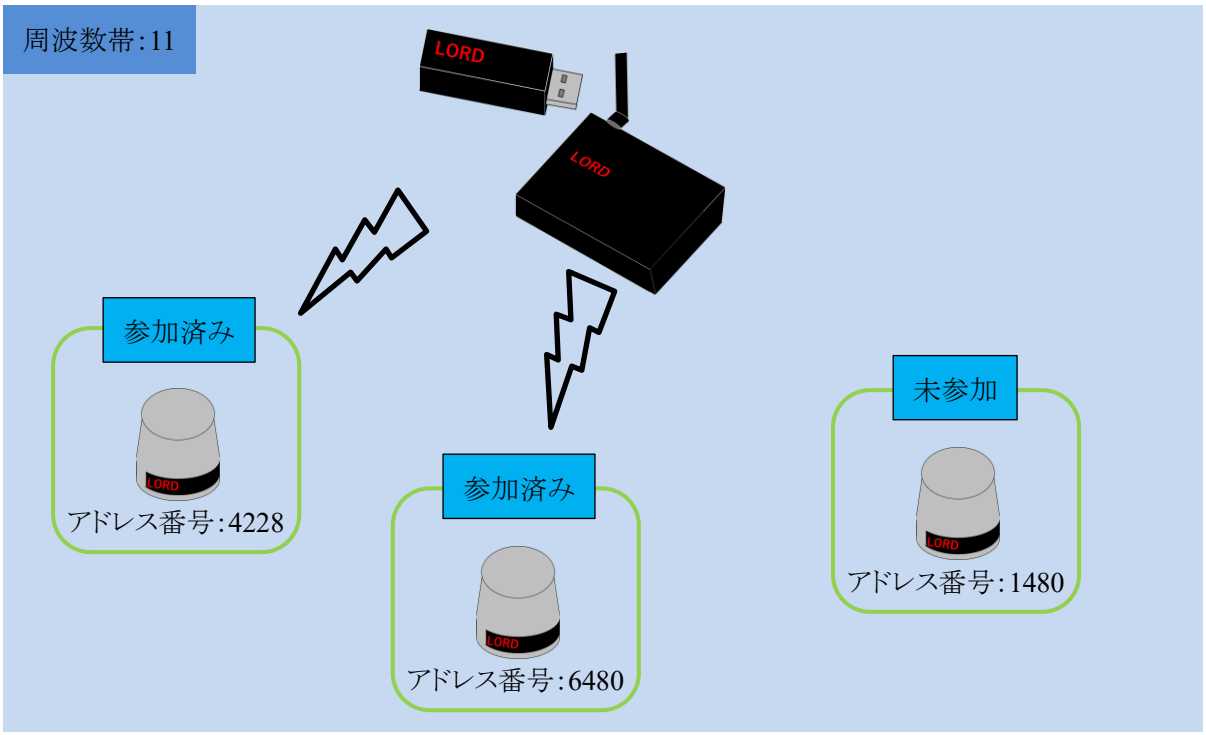
使用例(C#)

```
// EntrySampling コマンド実行
bool? entry = extension.Execute("Entrysampling", "") as bool?;
```

3.3.2.4.1. 同期サンプリングのグループについて

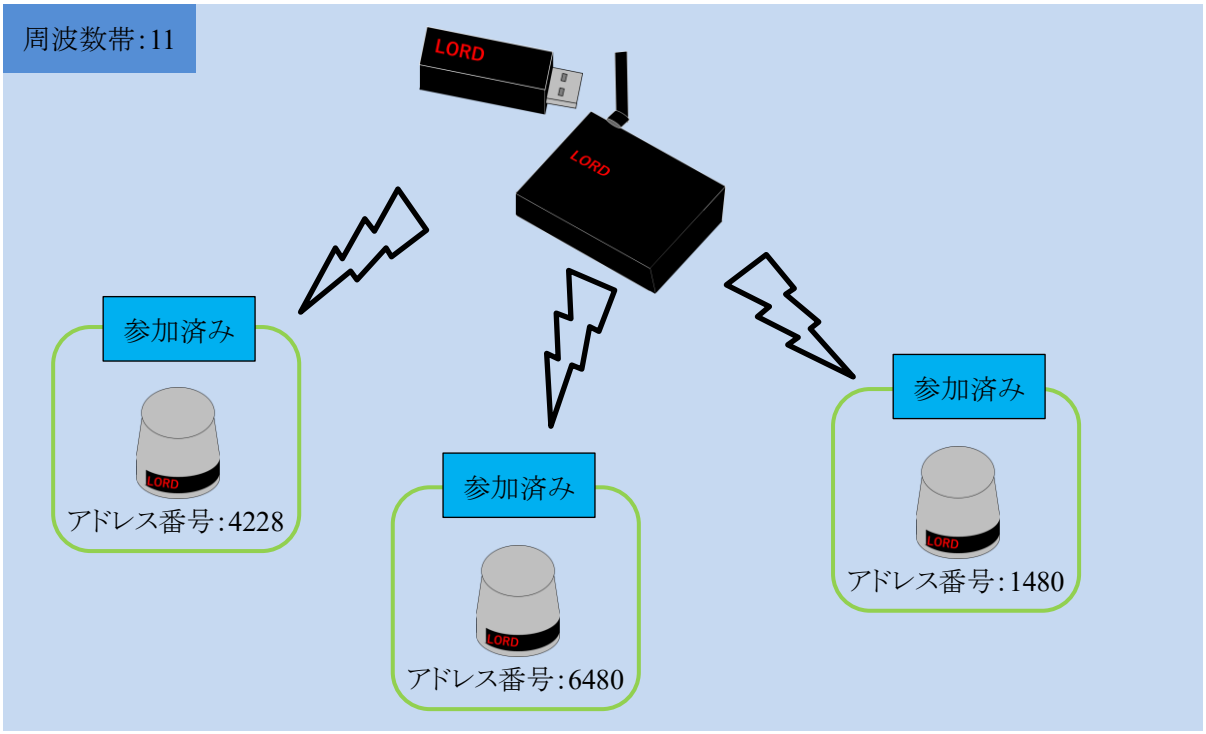
以下に同期サンプリングのグループについて詳細を示します．同期サンプリングのグループ未参加時に同期サンプリングを開始させた場合，または本コマンドを実行し，同期サンプリングのグループ参加後に同期サンプリング開始させた場合について説明します．ここではノードアドレス番号 1480 のノードを同期サンプリングのグループに参加させます．

・ノード 1480 が同期サンプリングのグループに参加前



上記図の状態で，Controller クラスの SyncStartSampling コマンドを実行した場合，ノードアドレス番号 4228 とノードアドレス番号 6480 のノードを使用して，同期サンプリングを開始させます．ノードアドレス番号 1480 のノードは同期サンプリングに含まれません．

・ノード 1480 が同期サンプリングのグループに参加後



ノードアドレス番号 1480 を EntrySampling コマンド使用して，同期サンプリングのグループに参加させます．参加後に上記の図の状態で，Controller クラスの SyncStartSampling コマンドを実行した場合，ノードアドレス番号 4228，6480，1480 のノードを使用して，同期サンプリングを開始させます．

3.4. 変数一覧

各クラスで使用可能な変数一覧を定義します. なお変数は, CaoVariable クラスのオブジェクトを指します.

3.4.1. システム変数とユーザー変数

プロバイダにはシステム変数とユーザー変数という 2 種類の変数が存在します.

システム変数

その変数を保持するオブジェクト内で唯一の情報にアクセスするための変数です. システム変数はしばしば静的データである場合があります. システム変数は名前の先頭に"@"がついています.

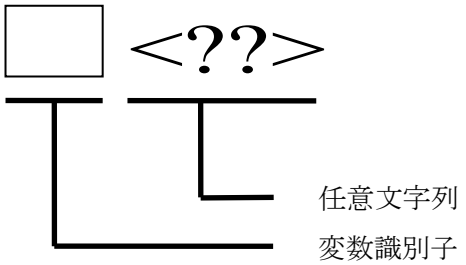
例)プロバイダバージョン, デバイス製造元, シリアル番号

ユーザー変数

変数を作成する際にどのような情報にアクセスするのかを, オプション文字列を使用することでユーザーが指定できる変数です. ユーザー変数はその性質上, 複数変数を登録(オプションのみ変更したい場合等に有用)するために変数識別の後に任意の文字列を付与することが可能です.

変数名に任意文字列を付与するための書式を以下に示します.

複数変数共通指定書式



3.4.2. CaoController クラス変数

変数名	説明	Value		参照
		get	put	
@MAKER_NAME	メーカー名を取得します.	○	-	P.30
@VERSION	プロバイダバージョンを取得します.	○	-	P.30
@NETWORKCAPACITY	同期サンプリングで使用する現在のネットワーク容量を取得します.	○	-	P.30
@COMMPROTOCOL	ゲートウェイの通信プロトコルを取得します.	○	-	P.31
@SERIALNO	ゲートウェイのシリアル番号を取得します.	○	-	P.31
@CLOUD_NAME	SenserCloud に使用する際のユニバーサルゲートウェイ名を取得します.	○	-	P.32

@FIRMWARE_VERSION	ゲートウェイのファームウェアバージョンを取得します.	○	-	P.32
@MODEL	ゲートウェイの機種モデルを取得します.	○	-	P.32

3.4.2.1. @MAKER_NAME

メーカー名の取得をします.

データ型

型説明	
VT_BSTR	メーカー名を取得します.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@MAKER_NAME", "");
// 値取得
string value = var.Value as string;
```

3.4.2.2. @VERSION

EXE のバージョンの取得をします.

データ型

型説明	
VT_BSTR	EXE のバージョンを取得します. *.* • .*

•

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@VERSION","");
// 値取得
string value = var.Value as string;
```

3.4.2.3. @NETWORKCAPACITY

同期サンプリングを実行する際の現在のネットワーク使用量をパーセンテージで取得します. CaoController クラスが保持している CaoExtension クラスに対応したノードの情報を元にネットワークの容量を取得します. この値が 100%より大きくなると SyncStartSampling コマンドを実行することができません. ネットワーク容量を変更させるには, AddExtension 時に Sampling オプションの値を指定するか, Extension クラスの @SAMPLING 変数にて値を変更し, サンプリング周期を変更する必要があります. また保持する CaoExtension クラスに対応したノードの個数・ノードの有効チャネル数も影響します.

データ型

型説明	
VT_R4	現在のネットワーク使用量を取得します.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@NETWORKCAPACITY ","");
// 値取得
float? value = var.Value as float;
```

3.4.2.4. @COMMPROTOCOL

ゲートウェイの通信プロトコルを取得します.

データ型

型説明	
VT_UI1	通信プロトコルを取得します. 以下の値が取得されます. 0 : LXRS 1 : LXRS+

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@COMMPROTOCOL", "");
// 値取得
Byte? commProtocol = var.Value as Byte?;
```

3.4.2.5. @SERIALNO

ゲートウェイのシリアル番号を取得します.

データ型

型説明	
VT_BSTR	シリアル番号を取得します.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@SERIALNO","");
// 値取得
string value = var.Value as string;
```

3.4.2.6. @CLOUD_NAME

Parker 社製のクラウドアプリケーション「SensorCloud」にて使用する際のユニバーサルベースステーション名を取得します.

データ型

型説明	
VT_BSTR	ユニバーサルベースステーション名を取得します.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@CLOUD_NAME","");
// 値取得
string value = var.Value as string;
```

3.4.2.7. @FIRMWARE_VERSION

ゲートウェイのファームウェアバージョンの取得をします.

データ型

型説明	
VT_BSTR	ファームウェアのバージョンを取得します. *.*.*

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@FIRMWARE_VERSION","");
// 値取得
string value = var.Value as string;
```

3.4.2.8. @MODEL

ゲートウェイのモデルを取得します.

データ型

型説明	
VT_BSTR	機種モデルを取得します. 値の詳細は, MSCL のドキュメントを参照してください.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@MODEL","");
// 値取得
```

```
string value = var.Value as string;
```

3.4.3. CaoExtension クラス変数

変数名	説明	Value		参照
		get	put	
@COMMPROTOCOL	ノードの通信プロトコルを取得します.	○	-	P.34
@SERIALNO	ノードのシリアル番号を取得します.	○	-	P.35
@CLOUD_NAME	SenserCloud に使用する際のユニバーサルセンサー名を取得します.	○	-	P.35
@FIRMWARE_VERSION	ノードのファームウェアバージョンを取得します.	○	-	P.35
@MODEL	ノードの機種モデルを取得します.	○	-	P.36
@SAMPLING	ノードのサンプリング周期を取得/設定します.	○	○	P.36
@DIAGNOSTIC_INFO	ノードから送られてくる診断情報を取得します.	○	-	P.37
@LASTSTATUS	ノードの最後のステータスを取得します.	○	-	P.39
@ENTRY	ノードの同期サンプリングのグループ参加状態を取得します.	○	-	P.39
COLLECTDATA<??>	チャネルデータを取得します.	○	-	P.40

※ ノードとの接続が一度でも成功した場合，接続時の値が取得されます.

ノードの状態によって取得可否が変わります. ノードの各状態の実行可能状態を以下に記述します.

変数名	実行可否		
	スリープ状態	アイドル状態	サンプリング状態
@COMMPROTOCOL	×	○	○
@SERIALNO	×	○	○
@CLOUD_NAME	○	○	○
@FIRMWARE_VERSION	×	○	○
@MODEL	×	○	○
@SAMPLING	×	○	×
@DIAGNOSTIC_INFO	○	○	○
@LASTSTATUS	○	○	○
COLLECTDATA<??>	○	○	○

3.4.3.1. @COMMPROTOCOL

ノードの通信プロトコルを取得します.

データ型

型説明

VT_UI1	通信プロトコルを取得します. 以下の値が取得されます. 0 : LXRS 1 : LXRS+

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = extension.AddVariable("@COMMPROTOCOL", "");
// 値取得
Byte? value= var.Value as Byte?;
```

3.4.3.2. @SERIALNO

ノードのシリアル番号を取得します.

データ型

型説明	
VT_BSTR	シリアル番号を取得します.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = extension.AddVariable("@SERIALNO", "");
// 値取得
string serialNo = var.Value as string;
```

3.4.3.3. @CLOUD_NAME

Parker 社製のクラウドアプリケーション「SensorCloud」にて使用する際のユニバーサルセンサー名を取得します.

データ型

型説明	
VT_BSTR	ユニバーサルセンサー名を取得します.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = extension.AddVariable("@CLOUD_NAME", "");
// 値取得
string cloudName = var.Value as string;
```

3.4.3.4. @FIRMWARE_VERSION

ノードのファームウェアバージョンの取得をします.

データ型

型説明	
VT_BSTR	ファームウェアのバージョンを取得します. *.*.*

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = extension.AddVariable("@FIRMWARE_VERSION", "");
// 値取得
string firmwareVersion = var.Value as string;
```

3.4.3.5. @MODEL

ノードのモデルを取得します.

データ型

型説明	
VT_BSTR	機種モデルを取得します. 値の詳細は, MSCL のドキュメントを参照してください.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = extension.AddVariable("@MODEL", "");
// 値取得
string model = var.Value as string;
```

3.4.3.6. @SAMPLING

同期サンプリング時のノードのサンプリング周期を取得/設定します. 実際にサンプリング周期がノードに反映されるのは 3.3.1.2.SyncStartSampling コマンド実行時になります.

データ型

型説明	
VT_I4	サンプリング周期. 値の詳細は表 3-2 を参照してください.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable varSampling = extension.AddVariable("@SAMPLING", "");
// 値取得
Int32? sampling = varSampling.Value as Int32?;
```

```
// 値を設定
varSampling.Value = 105;
```

3.4.3.7. @DIAGNOSTIC_INFO

ノードがアイドル状態の場合，定期的に診断情報が送信されます．WirelessMSCL プロバイダでは送信された最新の診断情報のみを保持し，本変数では保持している診断情報を取得します．

データ型

型説明		
VT_ARRAY | VT_VARIANT		診断情報
0	VT_UI1	現在のノードの状態
1	VT_UI4	アイドル実行時間(秒)
2	VT_UI4	スリープ実行時間（秒）
3	VT_UI4	アクティブな実行時間（秒）
4	VT_UI4	非アクティブな実行時間（秒）
5	VT_UI2	リセットカウンター
6	VT_UI4	組み込みテスト結果
7	VT_R4	内部温度(℃)
8	VT_UI1	バッテリー低下指標
9	VT_UI1	外部電源
10	VT_UI4	スイープインデックス
11	VT_UI4	悪いスイープ数
12	VT_UI4	総伝送数
13	VT_UI4	総再送数
14	VT_UI4	総ドロップパケット数
15	VT_UI4	同期試行数
16	VT_UI4	同期失敗数
17	VT_UI4	最後の同期からの経過時間(秒)
18	VT_UI4	イベントトリガーインデックス
19	VT_R4	データロギングメモリ状態(%)

※ 送信されたデータに該当の項目がない場合，VT_EMPTY となります．

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = extension.AddVariable("@DIAGNOSTIC_INFO", "");
```

```
// 値取得
object[] diagnosticInfo = var.Value as object[];

if (diagnosticInfo != null)
{
    // 現在のノードの状態
    Byte? status = diagnosticInfo[0] as Byte?;
    // アイドル実行時間(秒)
    UInt32? idleRuntime = diagnosticInfo[1] as UInt32?;
    // スリープ実行時間（秒）
    UInt32? sleepRuntime = diagnosticInfo[2] as UInt32?;
    // アクティブな実行時間（秒）
    UInt32? activeRuntime = diagnosticInfo[3] as UInt32?;
    // 非アクティブな実行時間（秒）
    UInt32? nonActiveRuntime = diagnosticInfo[4] as UInt32?;
    // リセットカウンター
    UInt16? resetCounter = diagnosticInfo[5] as UInt16?;
    // 組み込みテスト結果
    UInt32? testResult = diagnosticInfo[6] as UInt32?;
    // 内部温度(℃)
    float? inTemp = diagnosticInfo[7] as float?
    // バッテリー低下指標
    Byte? lowBattery = diagnosticInfo[8] as Byte?;
    // 外部電源
    Byte? extrnalPower = diagnosticInfo[9] as Byte?;
    // スイープインデックス
    UInt32? sweepIndex = diagnosticInfo[10] as UInt32?;
    // 悪いスイープ数
    UInt32? badSweepCount = diagnosticInfo[11] as UInt32?;
    // 総伝送数
    UInt32? totalSendCount = diagnosticInfo[12] as UInt32?;
    // 総再送数
    UInt32? totalReSendCount = diagnosticInfo[13] as UInt32?;
    // 総ドロップパケット数
    UInt32? totalDropPacketCount = diagnosticInfo[14] as UInt32?;
    // 同期試行数
    UInt32? syncTryCount = diagnosticInfo[15] as UInt32?;
```

```
        // 同期失敗数
        UInt32? syncFaildCount = diagnosticInfo[16] as UInt32?;
        // 最後の同期からの経過時間(秒)
        UInt32? lastSyncTime = diagnosticInfo[17] as UInt32?;
        // イベントトリガーインデックス(秒)
        UInt32? eventTriggerIndex = diagnosticInfo[18] as UInt32?;
        // データロギングメモリ状態(%)
        float? memmory = diagnosticInfo[19] as float?;
    }
```

3.4.3.8. @LASTSTATUS

ノードとの最後の接続時のステータスを取得します.

データ型

型	説明
VT_UI1	ノードのステータス. ステータスの詳細は以下の通りです. 0 : アイドル状態 1 : スリープ状態 2 : サンプリング状態 3 : サンプリング状態だが, ビーコンを失った状態 4 : サンプリング状態だが, アクティビティではない場合(アクティビティ検出モードの場合) 255 : 不明

使用例(C#)

```
    // 変数追加
    ORiN2.ManagedCAO.CCaoVariable var = extension.AddVariable("@LASTSTATUS", "");
    // 値取得
    int? lastStatus = var.Value as int?;
```

3.4.3.9. @ENTRY

ノードの同期サンプリンググループへの参加状態を取得します. この値が true の場合, Controller クラスの SyncStartSampling コマンドを実行すると, ノードの同期サンプリングが開始されます. false の場合同期サンプリングは開始されません. Extension クラスの EntrySampling コマンドを実行することによって, 同期サンプリンググループへの参加が可能です.

データ型

型	説明

VT_BOOL	同期サンプリンググループへの参加状態. true : 参加済み false : 未参加

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = extension.AddVariable("@ENTRY", "");
// 値取得
bool? entry = var.Value as bool?;
```

3.4.3.10. COLLECTDATA<??>

定期サンプリングにて蓄積しているチャネルデータを取り出します. COLLECTDATA の後に任意の文字列を入力して変数名を指定してください.

オプション

オプション	必須	説明	値範囲	デフォルト値
ChannelNo	○	チャネルデータのチャネル番号を指定します.	1 - 16	-----

データ型

型説明			
VT_ARRAY | VT_VARIANT			蓄積データ
0	VT_ARRAY | VT_BSTR		蓄積されたタイムスタンプ
	i	VT_BSTR	各チャネルデータに対応したタイムスタンプ
1	VT_ARRAY | VT_VARIANT		蓄積されたチャネルデータ
	i	VT_VARIANT	チャネルデータのデータ型によって取得されるデータが異なります.

※ 蓄積データがなければ VT_EMPTY が返却されます.

※ i : 蓄積されたデータ数分

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = extension.AddVariable("COLLECTDATA1", "ChannelNo=1");
// 値取得
object collectValues = var.Value as object;

// 値分解
```

```
if (collectValues != null)
{
    object[] values = collectValues as object[];
    // タイムスタンプ部分
    string[] timestamps = values[0] as string[];

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

    // 蓄積データ部分
    object[] collectDatas = values[1] as object[];

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

3.4.3.10.1. チャネルデータのデータ型について

取得したチャネルデータは以下のデータとして格納されます.

・**データ型が float の場合**

データ型
VT_R4

・**データ型が double の場合**

データ型
VT_R8

・**データ型が uint8 の場合**

データ型
VT_UI1

・**データ型が uint16 の場合**

データ型
VT_UI2

・**データ型が uint32 の場合**

データ型
VT_UI4

・データ型が **int8** の場合

データ型
VT_I1

・データ型が **int8** の場合

データ型
VT_I1

・データ型が **int16** の場合

データ型
VT_I2

・データ型が **int32** の場合

データ型
VT_I4

・データ型が **bool** の場合

データ型
VT_BOOL

・データ型が **float** の場合

データ型
VT_R4

・データ型が **string** または **Timestamp** の場合

データ型
VT_BSTR

・データ型が **Bytes** の場合

データ型
VT_ARRAY | VT_UI1

・データ型が **Vector** の場合**(1** 次元配列のデータ**)**

データ型
VT_ARRAY | VT_R4 or VT_ARRAY | VT_R8 or VT_ARRAY | VT_UI2 or VT_ARRAY | VT_UI1

・データ型が **Matrix** の場合**(2** 次元配列のデータ**)**

データ型
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 : 取得するデータによって異なります.

・データ型が ChannelMask の場合

データ型

VT_ARRAY | VT_UI4	
i	有効なチャネル番号

※ i : 有効なチャネル数分

・データ型が RfSweep の場合

データ型

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

※ i : 組み合わせ数分

・データ型が StructuralHealth の場合

データ型

VT_ARRAY | VT_VARIANT				
0	VT_R4			測定されている角度
1	VT_UI4			稼働時間カウンター
2	VT_R4			発生したダメージ (0% = 損傷無し, 100% = 死亡)
3	VT_ARRAY | VT_UI4			サンプルレート
	0	サンプルレートのタイプ		
	1	サンプルレートのサンプル数		
4	VT_ARRAY | VT_VARAINT			ヒストグラム
	i	VT_ARRAY | VT_VARIANT		ビン
		0	VT_VARIANT	ビンの開始範囲
		1	VT_VARIANT	ビンの終了範囲

		2	VT_VARIANT	ビンの範囲内にある値の数

※ i：組み合わせ数分

3.5. イベント一覧

コントローラのエラー通知や状態の変化を OnMessage イベントとして受け取ることが可能です.

番号	説明
0	データメモリ容量オーバーメッセージ

3.5.1.1. データメモリ容量オーバーメッセージ

定期サンプリングにてデータを取得した際に，プロバイダが使用しているメモリ量が，閾値を超えた際に発生するイベントです．本イベントが発生した場合，古い格納データを出力し，新規で取得されたデータを格納します．本イベントが発生する条件としては，定期サンプリングにて蓄積するスピードが，3.4.3.10.COLLECTDATA<??>変数の取得スピードより速い場合に発生します.

本イベントを発生させない為には，ゲートウェイへ格納されるスピードを遅くするか，3.4.3.10.COLLECTDATA<??>変数の取得スピードを速くするか，もしくは両方を行ってください.

Value プロパティデータ型

型説明		
VT_ARRAY | VT_VARIANT		
0	VT_UI4	ノードアドレス番号
1	VT_UI1	チャネル番号
2	VT_ARRAY | VTVARIANT	古い蓄積チャネルデータ．データの詳細は 3.4.3.10 を参照してください.

使用例(C#)

```
/// <summary>
/// メッセージ受信時の処理メソッド
/// </summary>
/// <param name="errorData">エラーデータ</param>
private void OnReceivedError(object[] errorData)
{
    Int32? errorCode = errorData[0] as Int32?;
    String packet = errorData[1] as String;
}
```

4. WirelessMSCL プロバイダによるプログラミング

WirelessMSCL プロバイダでは，以下の手順でクライアント PC とゲートウェイを接続またサンプリングする対象のノードを追加することができます.

- CaoEngine の作成
- CaoWorkspace の作成
- CaoController の作成

ゲートウェイに接続した後は，CaoController の CaoExtension オブジェクトを生成することでノード情報にアクセス，もしくは，CaoController の CaoVariable オブジェクトを生成することで，ゲートウェイの情報にアクセスすることができます.

- CaoExtension の作成

CaoExtension の CaoVariable オブジェクトを作成することで，ノードの情報にアクセスすることができます.

4.1. 同期サンプリングを開始しノードのチャネルデータを取得するサンプルプログラミング

ここでは例として同期サンプリングを開始させ，ゲートウェイからノードのチャネルデータの値を取得するサンプルプログラムを示します. 表 4-1 にサンプルプログラムの要件を，図 4-1 にサンプルプログラムの流れをそれぞれ記述しています.

表 4-1 サンプルプログラムの要件

要件	説明
接続先	シリアル通信で接続する
	接続先 COM ポート番号は 5
	ノードアドレス番号は 4228
	チャネル番号は 1
処理内容	取得するチャネルデータを追加する
	ゲートウェイの同期サンプリングを開始させる
	チャネルデータを取得する

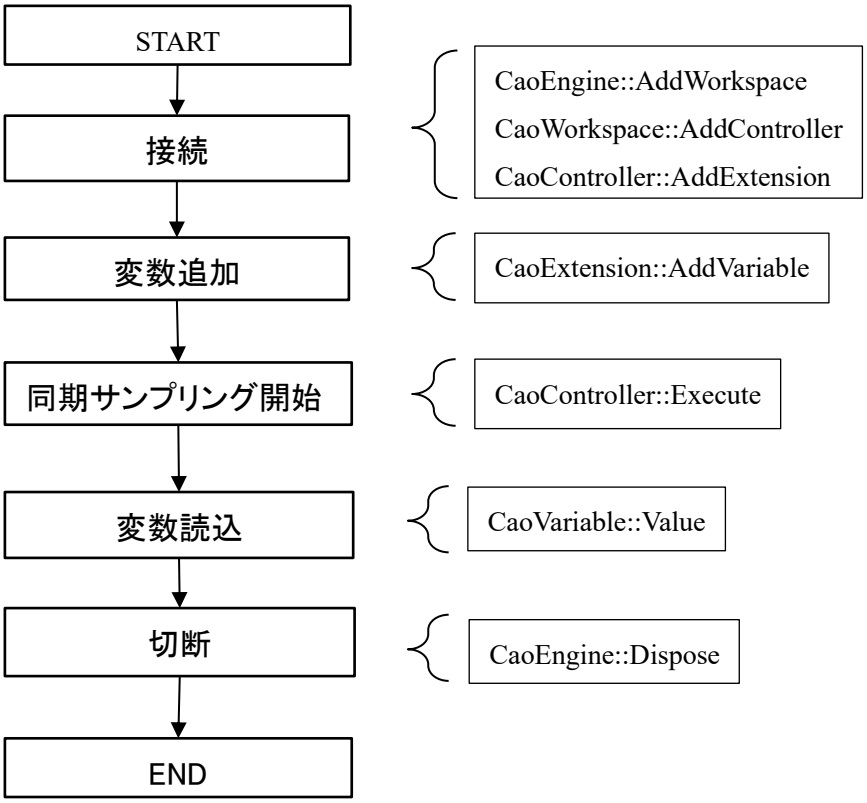


図 4-1 同期サンプリングを開始しノードのチャネルデータを取得する流れ

以降の節から具体的なコードを示します.

4.1.1. サンプルプログラム

以下にサンプルプログラムの全体像を示します.

Sample | Sample1.cs

```
// オブジェクト
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()
{
    // 接続
    this.Connect();

    // 同期サンプリング開始
    this.m_caoController.Execute("SyncStartSampling", null);

    // データ収集させるまで待機する
    System.Threading.Thread.Sleep(40);

    // 値取得
    object collectValues = this.m_varChannelData.Value as object;

    // 値分解
    if (collectValues != null)
    {
        object[] values = collectValues as object[];
        // タイムスタンプ部分
        string[] timestamps = values[0] as string[];

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

        // 蓄積データ部分
```

```
                object[] collectDatas = values[1] as object[];

                for (int i = 0; i < collectDatas.Length; i++)
                {
                    object collectData = collectDatas[i];
                }
            }
            // 切断
            this.Disconnect();
        }
        /// <summary>
        /// 接続メソッド
        /// </summary>
        private void Connect()
        {
            // CaoEngine オブジェクトの生成
            this.m_caoEngine = new ORiN2.ManagedCAO.CCaoEngine();
            // CaoWorkspace オブジェクトの生成
            this.m_caoWorkspace = this.m_caoEngine.AddWorkspace("NewWrks", "");
            // CaoController オブジェクトの生成
            this.m_caoController = this.m_caoWorkspace.AddController("WirelessGateway",
                                                                     "CaoProv.LORD.WirelessMSCL",
                                                                     "",
                                                                     "Conn=COM:5,DataCollectionCy
cle=30,IsBroadcastSetToIdle=true");
            // CaoExtension オブジェクトの作成
            this.m_caoExtension = this.m_caoController.AddExtension("Node4228",
"AddressNo=4228");
            // チャネルデータを Extension クラスに追加
            this.m_varChannelData = this.m_caoExtension.AddVariable("COLLECTDATA1",
"ChannelNo=1");
        }

        /// <summary>
        /// 切断メソッド
        /// </summary>
        private void Disconnect()
```

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

4.1.1.1. 接続

ゲートウェイとノードに接続するためには，以下の手順を取ります．

(1) オブジェクトを保持するための変数を用意します．

ゲートウェイ接続に必要なオブジェクトは，CaoEngine オブジェクトと CaoWorkspace オブジェクトと CaoController オブジェクトです．CaoWorkspace オブジェクトは，CaoController オブジェクトを CaoWorkspaces から取得する場合には変数を用意する必要はありません．また，ノードと接続するための CaoExtension オブジェクトが必要です．また変数にアクセスするための CaoVariable オブジェクトも必要になります．以下に C#でのコード例を示します．

使用例(C#)

```
// CaoEngine オブジェクト用の変数
private ORiN2.ManagedCAO.CCaoEngine m_caoEngine = null;
// CaoWorkspace オブジェクト用の変数
private ORiN2.ManagedCAO.CCaoWorkspace m_caoWorkspace = null;
// CaoController オブジェクト用の変数
private ORiN2.ManagedCAO.CCaoController m_caoController = null;
// CaoExtension オブジェクト用の変数
private ORiN2.ManagedCAO.CCaoExtension m_caoExtension = null;
// CaoVariable オブジェクト用の変数
private ORiN2.ManagedCAO.CCaoVariable m_varChannelData = null;
```

(2) CaoEngine オブジェクトを生成します．

CaoEngine オブジェクトは New キーワードを使って生成します．

使用例(C#)

```
// CaoEngine オブジェクトの生成
this.m_caoEngine = new ORiN2.ManagedCAO.CCaoEngine();
```

(3) CaoWorkspace オブジェクトを取得もしくは生成します．

CaoEngine オブジェクトを生成すると，デフォルトで CaoWorkspaces オブジェクトと CaoWorkspace オブジェクトを 1 つずつ生成しています．以下に CaoWorkspace オブジェクトを新しく生成するコード例とデフォルトの CaoWorkspace を示します．

使用例(C#)

```
// CaoWorkspace オブジェクトの生成
this.m_caoWorkspace = this.m_caoEngine.AddWorkspace("NewWrks", "");
```

(4) CaoController オブジェクトを生成します.

CaoController オブジェクトを生成するには, 使用するプロバイダ名と使用するためのパラメータを設定します. WirelessMSCL プロバイダでは, 接続先ゲートウェイの COM ポート番号, WirelessMSCL プロバイダがゲートウェイに定期的にデータ収集する周期, 接続時にゲートウェイと同周波数帯のノードをアイドル状態させるか否かをオプションで指定します. 以下にコード例を示します.

使用例(C#)

```
// CaoController オブジェクトの生成
this.m_caoController = this.m_caoWorkspace.AddController("WirelessGateway",
                                                          "CaoProv.LORD.WirelessMSCL",
                                                          "",
                                                          "Conn=COM:5,DataCollectionCy
cle=30,IsBroadcastSetToIdle=true");
```

(5) CaoExtension オブジェクトを生成します.

接続したいノードの CaoExtension オブジェクトを生成します. 以下にノードアドレス番号 4228 のノードにアクセスする Extension オブジェクトを生成するコード例を示します.

使用例(C#)

```
/ CaoExtension オブジェクトの作成
this.m_caoExtension = this.m_caoController.AddExtension("Node4228", "AddressNo=4228");
```

(6) CaoVariable オブジェクトを生成します.

取得したいチャネルデータの CaoVariable オブジェクトを作成します. CaoVariable オブジェクトを作成すると WirelessMSCL プロバイダがチャネルデータの蓄積を開始します. 以下に(5)にて作成したノードのチャネル番号 1 のチャネルデータにアクセスする Variable オブジェクトを生成するコードを示します.

使用例(C#)

```
// CaoVariable オブジェクトの作成
this.m_varChannelData = this.m_caoExtension.AddVariable("COLLECTDATA1",
 "ChannelNo=1");
```

4.1.1.2. 同期サンプリングの開始

接続したゲートウェイとノードを使用して, 同期サンプリングを開始させます. 同期サンプリングを開始させ

ることで，接続したノードからゲートウェイへデータが送られるようになります．WirelessMSCL プロバイダは 4.1.1.1 にて追加した，Variable オブジェクトにゲートウェイに蓄積されているデータを定期的に収集し，チャネルデータを蓄積するようになります．以下にコード例を示します．

使用例(C#)

```
// 同期サンプリング開始
this.m_caoController.Execute("SyncStartSampling", null);
```

4.1.1.3. チャネルデータの取得

蓄積したチャネルデータの値を取得するには，CaoVariable オブジェクトの Value プロパティを参照します．チャネルデータの値を取得する場合は，タイムスタンプ部分と蓄積データ部分に分割して取得します．以下にコード例を示します．

使用例(C#)

```
// 値取得
object collectValues = this.m_varChannelData.Value as object;

// 値分解
if (collectValues != null)
{
    object[] values = collectValues as object[];
    // タイムスタンプ部分
    string[] timestamps = values[0] as string[];

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

    // 蓄積データ部分
    object[] collectDatas = values[1] as object[];

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

4.1.1.4. 切断

コントローラと切断する場合には，生成したオブジェクトを消去すると共に，オブジェクトを管理するコレクションクラスから消去するオブジェクトを削除します．ただし，ORiN.ManagedCAO を使用した場合は明示的に削除する必要はありません．以下にコード例を示します．

使用例(C#)

```
// CaoEngine からすべてのオブジェクトを削除
this.m_caoEngine.Dispose();
// CaoEngine の消去
this.m_caoEngine = null;
```

4.2. ゲートウェイから同期サンプリングのネットワーク容量を取得するサンプルプログラミング

ここでは例としてゲートウェイから，同期サンプリングのネットワーク容量の値を取得するサンプルプログラムを示します． 表 4-2 にサンプルプログラムの要件を，図 4-2 にサンプルプログラムの流れをそれぞれ記述しています．

表 4-2 サンプルプログラムの要件

要件	説明
接続先	シリアル通信で接続する
	接続先 COM ポート番号は 5
	ノードアドレス番号は 4228 と 6480
処理内容	接続後にネットワーク容量を取得
	各ノードのサンプリング周期を設定
	再度ネットワーク容量を取得

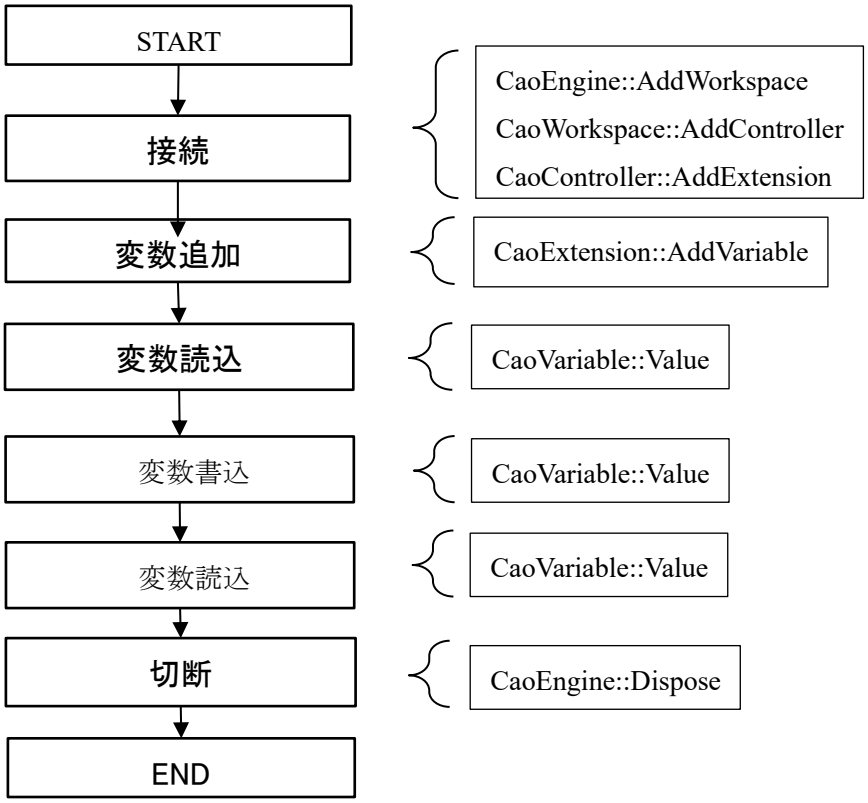


図 4-2 ゲートウェイから同期サンプリングのネットワーク容量を取得する流れ

以降の節から具体的なコードを示します.

4.2.1. サンプルプログラム

以下にサンプルプログラムの全体像を示します.

Sample　Sample1.cs

```
// オブジェクト
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()
{
    // 接続
    this.Connect();

    // ネットワーク容量を取得
    float? value = this.m_varNetworkCapacity.Value as float?;

    // ノード 4228 のサンプリング周期を 2 秒周期に変更
    this.m_varSampling1.Value = 114;

    // ノード 6480 のサンプリング周期を 256Hz 周期に変更
    this.m_varSampling2.Value = 105;

    // ネットワーク容量を再取得
    value = this.m_varNetworkCapacity.Value as float?;

    // 切断
    this.Disconnect();
}

/// <summary>
///  接続メソッド
/// </summary>
```

```
    private void Connect()
    {
        // CaoEngine オブジェクトの生成
        this.m_caoEngine = new ORiN2.ManagedCAO.CCaoEngine();
        // CaoWorkspace オブジェクトの生成
        this.m_caoWorkspace = this.m_caoEngine.AddWorkspace("NewWrks", "");
        // CaoController オブジェクトの生成
        this.m_caoController = this.m_caoWorkspace.AddController("WirelessGateway",
                                                                  "CaoProv.LORD.WirelessMSCL",
                                                                  "",
                               "Conn=COM:5,DataCollectionCycle=30,IsBroadcastSetToIdle=true");
        // ノード 4228 に対応した CaoExtension オブジェクトの作成
        this.m_extNode4228          =          this.m_caoController.AddExtension("Node4228",
"AddressNo=4228");
        // ノード 6480 に対応した CaoExtension オブジェクトの作成
        this.m_extNode6480          =          this.m_caoController.AddExtension("Node6480",
"AddressNo=6480");
        // ネットワーク容量変数を追加
        this.m_varNetworkCapacity = this.m_caoController.AddVariable("@NETWORKCAPACITY",
null);
        // ノードアドレス番号 4228 のサンプリング周期に対応した CaoVariable オブジェクトの作成
        this.m_varSampling1 = this.m_extNode4228.AddVariable("@SAMPLING", null);
        // ノードアドレス番号 6480 のサンプリング周期に対応した CaoVariable オブジェクトの作成
        this.m_varSampling2 = this.m_extNode6480.AddVariable("@SAMPLING", null);
    }

    /// <summary>
    ///  切断メソッド
    /// </summary>
    private void Disconnect()
    {
        this.m_caoEngine.Dispose();
        this.m_caoEngine = null;
    }
```

4.2.1.1. 接続

ゲートウェイとノードに接続するためには，以下の手順を取ります．

(1) オブジェクトを保持するための変数を用意します．

ゲートウェイ接続に必要なオブジェクトは，CaoEngine オブジェクトと CaoWorkspace オブジェクトと CaoController オブジェクトです．CaoWorkspace オブジェクトは，CaoController オブジェクトを CaoWorkspaces から取得する場合には変数を用意する必要はありません．また，ノードと接続するための CaoExtension オブジェクトが必要です．また変数にアクセスするための CaoVariable オブジェクトも必要になります．以下に C#でのコード例を示します．

使用例(C#)

```
// CaoEngine オブジェクト用の変数
private ORiN2.ManagedCAO.CCaoEngine m_caoEngine = null;
// CaoWorkspace オブジェクト用の変数
private ORiN2.ManagedCAO.CCaoWorkspace m_caoWorkspace = null;
// CaoController オブジェクト用の変数
private ORiN2.ManagedCAO.CCaoController m_caoController = null;
// CaoExtension オブジェクト用の変数
private ORiN2.ManagedCAO.CCaoExtension m_extNode4228 = null;
private ORiN2.ManagedCAO.CCaoExtension m_extNode6480 = null;
// CaoVariable オブジェクト用の変数
private ORiN2.ManagedCAO.CCaoVariable m_varNetworkCapacity = null;
private ORiN2.ManagedCAO.CCaoVariable m_varSampling1 = null;
private ORiN2.ManagedCAO.CCaoVariable m_varSampling2 = null;
```

(2) CaoEngine オブジェクトを生成します．

CaoEngine オブジェクトは New キーワードを使って生成します．

使用例(C#)

```
// CaoEngine オブジェクトの生成
this.m_caoEngine = new ORiN2.ManagedCAO.CCaoEngine();
```

(3) CaoWorkspace オブジェクトを取得もしくは生成します．

CaoEngine オブジェクトを生成すると，デフォルトで CaoWorkspaces オブジェクトと CaoWorkspace オブジェクトを 1 つずつ生成しています．以下に CaoWorkspace オブジェクトを新しく生成するコード例とデフォルトの CaoWorkspace を示します．

使用例(C#)

```
// CaoWorkspace オブジェクトの生成
```

```
this.m_caoWorkspace = this.m_caoEngine.AddWorkspace("NewWrks", "");
```

(4) CaoController オブジェクトを生成します.

CaoController オブジェクトを生成するには, 使用するプロバイダ名と使用するためのパラメータを設定します. WirelessMSCL プロバイダでは, 接続先ゲートウェイの COM ポート番号, WirelessMSCL プロバイダがゲートウェイに定期的にデータ収集する周期, 接続時にゲートウェイと同周波数帯のノードをアイドル状態させるか否かをオプションで指定します. 以下にコード例を示します.

使用例(C#)

```
// CaoController オブジェクトの生成
this.m_caoController = this.m_caoWorkspace.AddController("WirelessGateway",
                                                          "CaoProv.LORD.WirelessMSCL",
                                                          "",
                                                          "Conn=COM:5,DataCollectionCy
cle=30,IsBroadcastSetToIdle=true");
```

(5) CaoExtension オブジェクトを生成します.

接続したいノードの CaoExtension オブジェクトを生成します. 以下にノードアドレス番号 4228 のノードとノードアドレス番号 6480 にアクセスする Extension オブジェクトを生成するコード例を示します.

使用例(C#)

```
// ノード 4228 に対応した CaoExtension オブジェクトの作成
this.m_extNode4228 = this.m_caoController.AddExtension("Node4228", "AddressNo=4228");
// ノード 6480 に対応した CaoExtension オブジェクトの作成
this.m_extNode6480 = this.m_caoController.AddExtension("Node6480", "AddressNo=6480");
```

(6) CaoVariable オブジェクトを生成します.

取得するネットワーク容量と, 各ノードのサンプリング周期を設定する CaoVariable オブジェクトを作成します. 以下にネットワーク容量とサンプリング周期にアクセスする Variable オブジェクトを生成するコードを示します.

使用例(C#)

```
// ネットワーク容量の CaoVariable オブジェクトの作成
this.m_varNetworkCapacity = this.m_caoController.AddVariable("@NETWORKCAPACITY",
null);
// ノードアドレス番号 4228 のサンプリング周期に対応した CaoVariable オブジェクトの作成
this.m_varSampling1 = this.m_extNode4228.AddVariable("@SAMPLING", null);
// ノードアドレス番号 6480 のサンプリング周期に対応した CaoVariable オブジェクトの作成
this.m_varSampling2 = this.m_extNode6480.AddVariable("@SAMPLING", null);
```

4.2.1.2. 接続後のネットワーク容量の取得

接続したゲートウェイとノードの同期サンプリングのネットワーク容量を取得します．以下にコード例を示します．

使用例(C#)

```
// ネットワーク容量を取得
float? value = this.m_varNetworkCapacity.Value as float?;
```

4.2.1.3. 各ノードのサンプリング周期を設定

各ノードの同期サンプリング実行時のサンプリング周期を設定します．このサンプリング周期と接続ノード数によってネットワーク容量が決まります．以下にコード例を示します．

使用例(C#)

```
// ノード 4228 のサンプリング周期を 2 秒周期に変更
this.m_varSampling1.Value = 114;
// ノード 6480 のサンプリング周期を 256Hz 周期に変更
this.m_varSampling2.Value = 105;
```

4.2.1.4. サンプリング周期設定後のネットワーク容量を取得

各ノードのサンプリング周期を設定した後に，再度ネットワーク容量を取得します．以下にコード例を示します．

使用例(C#)

```
// ネットワーク容量を再取得
value = this.m_varNetworkCapacity.Value as float?;
```

4.2.1.5. 切断

コントローラと切断する場合には，生成したオブジェクトを消去すると共に，オブジェクトを管理するコレクションクラスから消去するオブジェクトを削除します．ただし，ORiN.ManagedCAO を使用した場合は明示的に削除する必要はありません．以下にコード例を示します．

使用例(C#)

```
// CaoEngine からすべてのオブジェクトを削除
this.m_caoEngine.Dispose();
// CaoEngine の消去
this.m_caoEngine = null;
```

4.3. ノードの同期サンプリングを終了させるサンプルプログラミング

ここでは例として同期サンプリングを開始させ，その後同期サンプリングを明示的に終了させるサンプルプログラムを示します．表 4-3 にサンプルプログラムの要件を，図 4-3 にサンプルプログラムの流れをそれぞれ記述しています．

表 4-3 サンプルプログラムの要件

要件	説明
接続先	シリアル通信で接続する
	接続先 COM ポート番号は 5
	ノードアドレス番号は 4228 と 6480
処理内容	同期サンプリングを開始
	同期サンプリングを終了

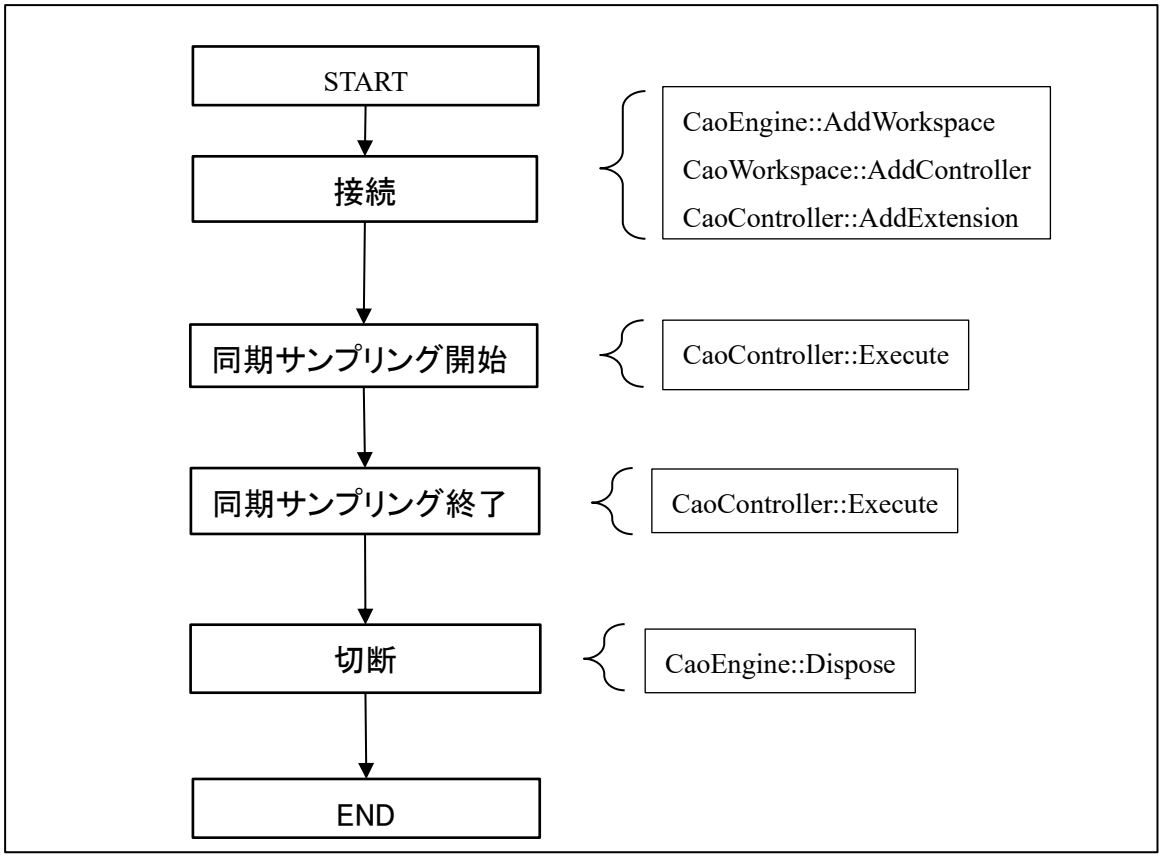


図 4-3 同期サンプリングを開始しノードのチャネルデータを取得する流れ

以降の節から具体的なコードを示します．

4.3.1. サンプルプログラム

以下にサンプルプログラムの全体像を示します.

Sample Sample1.cs

```
// オブジェクト
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()
{
    // 接続
    this.Connect();

    // 同期サンプリング開始
    object resultStatus = this.m_caoController.Execute("SyncStartSampling", null) as object;

    // 同期サンプリング実行後のノードステータス
    if (resultStatus != null)
    {
        if (resultStatus is ushort[])
        {
            ushort[] status = resultStatus as ushort[];
            for (int i = 0; i < status.Length; i++)
            {
                // 各ノードのステータス(アドレス番号昇順)
                ushort state = status[i];
            }
        }

    }

    // 同期サンプリング終了
    object nodeStatus = this.m_caoController.Execute("BroadcastSetToIdle", null) as object;

    // 同期サンプリング終了後のノードステータス
```

```
            if (nodeStatus != null)
            {
                if (nodeStatus is ushort[])
                {
                    ushort[] status = nodeStatus as ushort[];
                    for (int i = 0; i < status.Length; i++)
                    {
                        // 各ノードのステータス(アドレス番号昇順)
                        ushort state = status[i];
                    }
                }
            }
            // 切断
            this.Disconnect();
        }

        /// <summary>
        /// 接続メソッド
        /// </summary>
        private void Connect()
        {
            // CaoEngine オブジェクトの生成
            this.m_caoEngine = new ORiN2.ManagedCAO.CCaoEngine();
            // CaoWorkspace オブジェクトの生成
            this.m_caoWorkspace = this.m_caoEngine.AddWorkspace("NewWrks", "");
            // CaoController オブジェクトの生成
            this.m_caoController = this.m_caoWorkspace.AddController("WirelessGateway",
                                                                     "CaoProv.LORD.WirelessMSCL",
                                                                     "",
                                "Conn=COM:5,DataCollectionCycle=30,IsBroadcastSetToIdle=true");
            // ノード 4228 に対応した CaoExtension オブジェクトの作成
            this.m_extNode4228          =          this.m_caoController.AddExtension("Node4228",
"AddressNo=4228");
            // ノード 6480 に対応した CaoExtension オブジェクトの作成
            this.m_extNode6480          =          this.m_caoController.AddExtension("Node6480",
"AddressNo=6480");
        }
```

```
/// <summary>
/// 切断メソッド
/// </summary>
private void Disconnect()
{
    this.m_caoEngine.Dispose();
    this.m_caoEngine = null;
}
```

4.3.1.1. 接続

ゲートウェイとノードに接続するためには，以下の手順を取ります．

(1) オブジェクトを保持するための変数を用意します．

ゲートウェイ接続に必要なオブジェクトは，CaoEngine オブジェクトと CaoWorkspace オブジェクトと CaoController オブジェクトです．CaoWorkspace オブジェクトは，CaoController オブジェクトを CaoWorkspaces から取得する場合には変数を用意する必要はありません．また，ノードと接続するための CaoExtension オブジェクトが必要です．

使用例(C#)

```
// CaoEngine オブジェクト用の変数
private ORiN2.ManagedCAO.CCaoEngine m_caoEngine = null;
// CaoWorkspace オブジェクト用の変数
private ORiN2.ManagedCAO.CCaoWorkspace m_caoWorkspace = null;
// CaoController オブジェクト用の変数
private ORiN2.ManagedCAO.CCaoController m_caoController = null;
// CaoExtension オブジェクト用の変数
private ORiN2.ManagedCAO.CCaoExtension m_extNode4228 = null;
private ORiN2.ManagedCAO.CCaoExtension m_extNode6480 = null;
```

(2) CaoEngine オブジェクトを生成します．

CaoEngine オブジェクトは New キーワードを使って生成します．

使用例(C#)

```
// CaoEngine オブジェクトの生成
this.m_caoEngine = new ORiN2.ManagedCAO.CCaoEngine();
```

(3) CaoWorkspace オブジェクトを取得もしくは生成します．

CaoEngine オブジェクトを生成すると，デフォルトで CaoWorkspaces オブジェクトと CaoWorkspace オブジ

ェクトを 1 つずつ生成しています. 以下に CaoWorkspace オブジェクトを新しく生成するコード例とデフォルトの CaoWorkspace を示します.

使用例(C#)

```
// CaoWorkspace オブジェクトの生成
this.m_caoWorkspace = this.m_caoEngine.AddWorkspace("NewWrks", "");
```

(4) CaoController オブジェクトを生成します.

CaoController オブジェクトを生成するには, 使用するプロバイダ名と使用するためのパラメータを設定します. WirelessMSCL プロバイダでは, 接続先ゲートウェイの COM ポート番号, WirelessMSCL プロバイダがゲートウェイに定期的にデータ収集する周期, 接続時にゲートウェイと同周波数帯のノードをアイドル状態させるか否かをオプションで指定します. 以下にコード例を示します.

使用例(C#)

```
// CaoController オブジェクトの生成
this.m_caoController = this.m_caoWorkspace.AddController("WirelessGateway",
                                                          "CaoProv.LORD.WirelessMSCL",
                                                          "",
                                                          "Conn=COM:5,DataCollectionCy
cle=30,IsBroadcastSetToIdle=true");
```

(5) CaoExtension オブジェクトを生成します.

接続したいノードの CaoExtension オブジェクトを生成します. 以下にノードアドレス番号 4228 のノードとノードアドレス番号 6480 にアクセスする Extension オブジェクトを生成するコード例を示します.

使用例(C#)

```
// ノード 4228 に対応した CaoExtension オブジェクトの作成
this.m_extNode4228 = this.m_caoController.AddExtension("Node4228", "AddressNo=4228");
// ノード 6480 に対応した CaoExtension オブジェクトの作成
this.m_extNode6480 = this.m_caoController.AddExtension("Node6480", "AddressNo=6480");
```

4.3.1.2. 同期サンプリングの開始

接続したゲートウェイとノードを使用して, 同期サンプリングを開始させます. 同期サンプリングを開始させることで, 接続したノードからゲートウェイへデータが送られるようになります. 以下にコード例を示します.

使用例(C#)

```
// 同期サンプリング開始
object resultStatus = this.m_caoController.Execute("SyncStartSampling", null) as object;
// 同期サンプリング実行後のノードステータス
```

```
if (resultStatus != null)
{
    if (resultStatus is ushort[])
    {
        ushort[] status = resultStatus as ushort[];
        for (int i = 0; i < status.Length; i++)
        {
            // 各ノードのステータス(アドレス番号昇順)
            ushort state = status[i];
        }
    }
}
```

4.3.1.3. 同期サンプリングを終了

同期サンプリングを開始させた後に，明示的に同期サンプリングを終了させます．以下にコード例を示します．

使用例(C#)

```
// 同期サンプリング終了
object nodeStatus = this.m_caoController.Execute("BroadcastSetToIdle", null) as object;

// 同期サンプリング終了後のノードステータス
if (nodeStatus != null)
{
    if (nodeStatus is ushort[])
    {
        ushort[] status = nodeStatus as ushort[];
        for (int i = 0; i < status.Length; i++)
        {
            // 各ノードのステータス(アドレス番号昇順)
            ushort state = status[i];
        }
    }
}
```

4.3.1.4. 切断

コントローラと切断する場合には，生成したオブジェクトを消去すると共に，オブジェクトを管理するコレクショ

ンクラスから消去するオブジェクトを削除します．ただし，ORiN.ManagedCAO を使用した場合は明示的に削除する必要はありません．以下にコード例を示します．

使用例(C#)

```
// CaoEngine からすべてのオブジェクトを削除
this.m_caoEngine.Dispose();
// CaoEngine の消去
this.m_caoEngine = null;
```

5. WirelessMSCL プロバイダエラーコード

本プロバイダには，0x8011****でマスクした以下の独自エラーコードが存在します(表 5-1 独自エラーコード表参照). また，発生する箇所によってエラーコードのマスク値が変わります．CaoWorkspace クラスで発生するエラーコードは 0x80110***，CaoController クラスで発生するエラーコードは 0x80111***，CaoExtension クラスで発生するエラーコードは 0x80112***，CaoVariable クラスで発生するエラーコードは 0x80113***となっています．

ORiN2 の共通エラーについては，「ORiN2 プログラミングガイド」のエラーコードの章を参照してください．

表 5-1 独自エラーコード表

エラー番号	説明	対策
0x80110001	Conn オプションの指定が不正です.	3.2.1.1.1.Conn オプション を確認し，正しいオプションを指定してください.
0x80110002	Timeout オプションの指定が不正です.	3.2.1.1.AddController メソッドを確認し，正しいオプションを指定してください.
0x80110003	DataCollectionCycle オプションの指定が不正です.	3.2.1.1.AddController メソッドを確認し，正しいオプションを指定してください.
0x80110004	IsBroadcastSetToIdle オプションの指定が不正です.	3.2.1.1.AddController メソッドを確認し，正しいオプションを指定してください.
0x80110005	ゲートウェイとの接続に失敗しました.	ゲートウェイが接続されていない Com ポート番号を指定した可能性があります．正しい Com ポート番号を指定してください.
0x80110006	既に接続済みのゲートウェイに接続しようとし，失敗しました.	Controller クラスに存在しないゲートウェイを指定してください.
0x80111001	同期サンプリングを開始できませんでした.	機器の状態とノードの状態を確認してください.
0x80111002	同期サンプリング内のネットワーク使用量がオーバーしました.	100%以下になるように，ノード数，ノードのサンプリング周期を設定し直してください.
0x80111003	存在しないコマンド名を指定しました.	3.3.1.CaoController クラスコマンドに存在するコマンド名を指定してください.
0x80111011	存在しない変数名を指定しました.	3.4.2.CaoController クラス変数に存在する変数名を指定してください.
0x80111021	アドレス番号が未指定です.	3.2.2.3.AddExtension メソッドを確認し，正しいオプションを指定してください.
0x80111022	アドレス番号の値が範囲外です.	3.2.2.3.AddExtension メソッドを確認し，正しいオプションを指定してください.

エラー番号	説明	対策
0x80111023	既に存在するアドレス番号を指定しました.	Controller クラスに存在する Extension クラスのノードアドレス番号以外のアドレス番号を指定してください.
0x80111024	サンプリング周期の値が範囲外です.	3.2.2.3.AddExtension メソッドを確認し, 正しいオプションを指定してください.
0x80112001	存在しないコマンド名を指定しました.	3.3.2.CaoExtension クラスコマンドに存在するコマンド名を指定してください.
0x80112011	存在しない変数名を指定しました.	3.4.3.CaoExtension クラス変数に存在する変数名を指定してください.
0x80112012	チャネル番号が未指定です.	3.4.3.10.COLLECTDATA<??> を確認し, ChannelNo オプションを指定してください.
0x80112013	チャネル番号の値が範囲外です.	チャネル番号を範囲内の値で指定してください.
0x80112014	既に存在するチャネル番号を指定しました.	Extension クラスに存在する COLLECTDATA<??>変数のチャネル番号以外のチャネル番号を指定してください.
0x80113001	サンプリング周期の値が範囲外です.	表 3-2 に存在する値を指定してください.

また, 本プロバイダは, API で発生したエラーのエラーコードをそのまま返します.
API のエラー詳細は, MSCL のドキュメントに記述されていますのでそちらを参照してください.

付録A. API 対応表

CaoWorkspace::AddController

API 関数名
mscl.Connection.Serial
mscl.Devices.listBaseStations
mscl.BaseStation.ping
mscl.BaseStation.broadcastSetToIdle ※ CaoWorkspace::AddController のオプションで IsBroadcastSetToIdle=true を指定したとき

CaoController(定期サンプリング実行時)

API 関数名
mscl.BaseStation.getData
mscl.DataSweep.nodeAddress
mscl.DataSweep.timestamp
mscl.DataSweep.data
mscl.WirelessDataPoint.channelId
mscl.WirelessDataPoint.storedAs
mscl.WirelessDataPoint.as_bool ※ データ型が bool の場合
mscl.WirelessDataPoint.as_Bytes ※ データ型が Bytes の場合
mscl.WirelessDataPoint.as_ChannelMask mscl.ChannelMask.enabled ※ データ型が ChannelMask の場合
mscl.WirelessDataPoint.as_double ※ データ型が double の場合
mscl.WirelessDataPoint.as_float ※ データ型が float の場合
mscl.WirelessDataPoint.as_int16 ※ データ型が int16 の場合
mscl.WirelessDataPoint.as_int32 ※ データ型が int32 の場合
mscl.WirelessDataPoint.as_int8 ※ データ型が int8 の場合
mscl.WirelessDataPoint.as_Matrix

API 関数名
mscl.Matrix.valuesType mscl.Matrix.rows mscl.Matrix.columns mscl.Matrix.as_doubleAt ： Matrix 内部のデータ型が double の場合 mscl.Matrix.as_floatAt ： Matrix 内部のデータ型が float の場合 mscl.Matrix.as_uint16At ： Matrix 内部のデータ型が uint16 の場合 mscl.Matrix.as_uint8At ： Matrix 内部のデータ型が uint8 の場合 ※ データ型が Matrix の場合
mscl.WirelessDataPoint.as_RfSweep ※ データ型が RfSweep の場合
mscl.WirelessDataPoint.as_string ※ データ型が string の場合
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 ※ データ型が StructuralHealth の場合
mscl.WirelessDataPoint.as_Timestamp ※ データ型が Timestamp の場合
mscl.WirelessDataPoint.as_uint16 ※ データ型が uint16 の場合
mscl.WirelessDataPoint.as_uint32 ※ データ型が uint32 の場合
mscl.WirelessDataPoint.as_uint8 ※ データ型が uint8 の場合
mscl.WirelessDataPoint.as_Vector mscl.Vector.as_doubleAt ： Matrix 内部のデータ型が double の場合 mscl.Vector.as_floatAt ： Matrix 内部のデータ型が float の場合

API 関数名
mscl.Vector.as_uint16At ： Matrix 内部のデータ型が uint16 の場合 mscl.Vector.as_uint8At ： Matrix 内部のデータ型が uint8 の場合 ※ データ型が Vector の場合

CaoController::Execute

コマンド名	API 関数名
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 関数名
mscl.WirelessNodeConfig.dataCollectionMethod
mscl.WirelessNodeConfig.sampling
mscl.WirelessNodeConfig.sampleRate ※ CaoController::AddExtension のオプション Sampling を指定した場合
mscl.WirelessNode.applyConfig

CaoController::Execute

コマンド名	API 関数名
SetToIdle	mscl.WirelessNode.setToIdle mscl.WirelessNode.lastDeviceState
Ping	mscl.WirelessNode.ping
GetActiveChannels	mscl.WirelessNode.getActiveChannels

CaoVariable

親クラス	変数名	API 関数名
CaoController	@NETWORKCAPACITY	mscl.SyncSamplingNetwork.percentBandwidth
CaoController	@COMMPROTOCOL	mscl.BaseStation.communicationProtocol

親クラス	変数名	API 関数名
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	get_value 時: mscl.WirelessNode.getSampleRate put_value 時: mscl.WirelessNodeConfig.sampleRate mscl.WirelessNode.applyConfig
CaoExtension	@LASTSTATUS	mscl.WirelessNode.lastDeviceState

付録B. ノードとの接続時の注意点と対処法

ゲートウェイと接続時に，ノードがサンプリング状態またはスリープ状態の場合はノードと接続を行うことができません（同期サンプリングや，ノードの情報などの取得ができない）.

以下に，それぞれの対処法を記述いたします.

・ゲートウェイと接続時にノードがサンプリング状態の場合

3.2.1.1.AddController 実行時のオプションの IsBroadcastSetToIdle を true で指定するか，3.3.1.1.BroadcastSetToIdle コマンドを実行することで，サンプリング状態からアイドル状態になり接続が可能となります.

・ゲートウェイと接続時にノードがスリープ状態の場合

ゲートウェイと接続時にノードがスリープ状態場合，クライアント PC 等からは操作が不可能です．ですので，ノードのスイッチから物理的にスリープ状態からアイドル状態にすることで接続が可能となります.

付録C. OnMessage が発生するまでの時間目安

同期サンプリングを開始後に，蓄積データの取り出しを行わなかった場合に 3.5.1.1.データメモリ容量オーバーメッセージが発生する時間は，ゲートウェイの接続台数及びネットワーク使用量，サンプリング周期によって発生時間が異なります．以下に条件と時間の目安を示します．

- **共通環境**
  ゲートウェイ ： 1 台
  ノード ： 1 台

1. 通信プロトコルが LXRS の場合

ノードのサンプリング周期	周波数帯のネットワーク使用量	サンプリング開始から OnMessage が発生する時間
512Hz	100%	約 34 分
256Hz	50%	約 68 分
128Hz	25%	約 136 分

2. 通信プロトコルが LXRS+の場合

ノードのサンプリング周期	周波数帯のネットワーク使用量	サンプリング開始から OnMessage が発生する時間
2048Hz	100%	約 8 分 30 秒
1024Hz	50%	約 17 分
512Hz	25%	約 34 分