KEYENCE
LJ-X8000A プロバイダ
ユーザーズ ガイド

Version 1.0.0

July 16, 2020

備考：

【改版履歴】

バージョン	日付	内容
1.0.0	2020-07-16	初版.

【動作確認機種】

機種	バージョン	注意事項
LJ-X8080A		

目次

1. はじめに

本書は，株式会社 KEYENCE の LJ-X8000A シリーズに対してプロファイルデータの取得を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．以降本プロバイダを LJX8000A プロバイダと呼称します．

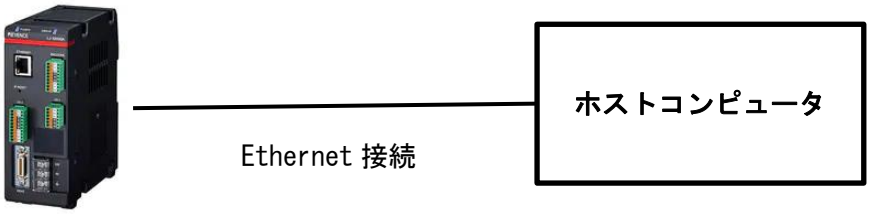


図 1-1 構成図

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．）

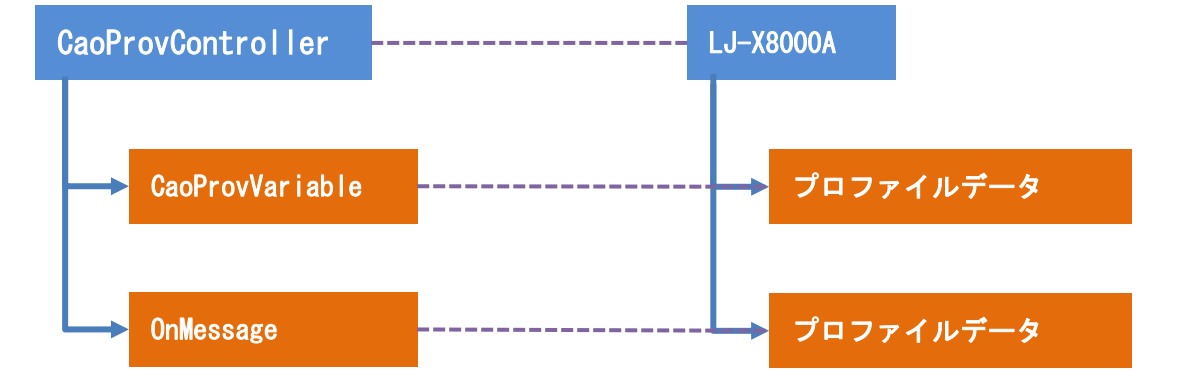


図 1-2 プロバイダの構成とデバイス情報との対応図

2. コマンドリファレンス

2.1. メソッド/プロパティー覧

表 2-1 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.8
CaoController				
	VariableNames	P	接続可能な変数名リストの取得	P.10
	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.10
	AddVariable	M	変数オブジェクトの追加	P.10
	Execute	M	拡張コマンドの実行	P.11
	OnMessage	E	メッセージ受信イベント	P.26
CaoVariable				
	Value	P	値の取得/設定	P.26

2.2. メソッド・プロパティ

2.2.1. CaoWorkspace クラス

2.2.1.1. 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. LJ-X8000A 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 LJ-X8000A コントローラと接続を行います. 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)
        "CaoProv.KEYENCE.LJ-X8000A",    // 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(省略可能)
)
```

[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.

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
Conn=<通信パラメータ>	○	接続先情報を指定します
DeviceID=<ID>	○	接続対象のコントローラに固有の ID を設定します. ID は 0~5 の範囲で設定し,複数接続する場合には重複の無いようにしてください 例 : "DeviceID=1"
HeadType=<ヘッド種別>	--	コントローラに接続するヘッドの種別を指定します. X:LJ-X ヘッド V:LJ-V ヘッド
HSDComm=<高速データ通信 ON/OFF>[:<高速データ通信用ポート番号>:<1 度にまとめて送信するプロファイル数>]	--	AddController 時に高速データ通信を開始するかどうかを指定します.高速データ通信 OFF(1 番目のパラメータが 0)の場合, 2 番目以降のパラメータを省略することができます. (デフォルト:0) <高速データ通信 ON/OFF>: 0: OFF 1: ON 例: "HSDComm=0" "HSDComm=1:24691:10"
HSDCommEndMess=	--	高速データ通信終了時にメッセージを送信するか指定します. 0:メッセージ送信なし 1:メッセージ送信あり

使用例

```
// 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("LJ-X8000A
                    "CaoProv.KEYENCE.LJ-X8000A",
                    "",
                    "Conn = ETH:192.168.10.10,DeviceId=1,HeadType=X,Timeout = 1000");
```

2.2.1.1.1. 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. ここで角括弧(″[]″)内は省略可能なことを,各パラメータの解説中の下線部はオプションを指定しなかった時のデフォルト値をそれぞれ示します.

TCP/IP で接続する場合

″Conn=ETH:<接続先 IP>[:<接続先ポート>[:<ローカル IP>[:<ローカルポート>]]]″

<接続先 IP> : 接続先 IP アドレスを***.***.***.***の形式で指定します.
この項目は必ず指定してください.

<接続先ポート> : 接続先ポート番号を指定します. 24691

2.2.1.1.2. HSDComm オプション注意事項

1 度にまとめて送信するプロファイル数は設定値によっては, メモリが不足する可能性があります.

動作確認環境では 3 万プロファイルを同時に受信できることを確認しておりますが実行環境により許容範囲は異なります.

2.2.2. CaoController クラス

2.2.2.1. VariableNames プロパティ

接続可能な変数名リストを取得します. 本プロパティで取得した変数名は, 後述する AddVariable メソッドの第一引数に使用することができます.

使用例

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

2.2.2.2. 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

```
// 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];
```

2.2.2.3. 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 2.3.1 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)
)
```

2.2.2.4. Execute メソッド

CaoController の拡張コマンドを実行します. 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)
)
```

以下に, 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
システム制御コマンド		
GetError	コントローラ内で発生しているエラーコードを, 指定した数だけ取得します.	P.12
ClearError	指定したエラーコードをクリアします.	P.13
TrgErrorReset	TRG_ERROR の ON 状態をクリアします.	P.13
GetTriggerAndPulseCount	トリガカウントおよびパルスカウントを取得します.	P.13
GetHeadTemperture	ヘッドの温度を取得します.	P.14
GetSerialNumber	シリアル番号を取得します.	P.14
GetAttentionStatus	TRG_ERROR/MEM_FULL/TRG_PASS の状態を取得します	P.15
GetLedLightImage	LED 照明画像を取得します.	P.15
測定制御コマンド		
Trigger	トリガーを発行します.	P.16
StartMesure	バッチ測定を開始します.	P.16
StopMesure	バッチ測定を終了します.	P.16

コマンド	説明	参照
ClearMemory	コントローラ内に蓄積されたデータをクリアします.	P.17
設定変更/読み出し関連コマンド		
GetActiveProgram	現在のアクティブプログラム No.を取得します.	P.17
ChangeActiveProgram	アクティブプログラム No.を切り替えます.	P.17
測定結果取得コマンド		
GetProfile	プロファイルデータを取得します	P.17
GetBatchProfile	バッチプロファイルデータを取得します.	P.19
GetbatchSimpleArray	バッチプロファイルデータ(SimpleArray)を取得します.	P.22
SetProfileCount	1 回の通信でプロファイルデータ数を設定します.	P.24
高速データ通信関連コマンド		
StartHighSpeedDataCommunication	高速データ通信を開始します.	P.25
StartHighSpeedDataCommunicationSimpleArray	高速データ通信(SimpleArray)を開始します.	P.26
StopHighSpeedDataCommunication	高速データ通信を終了します.	P.26

2.2.2.4.1. GetError コマンド

エラー件数と指定した数のエラーコードを取得します.引数として 0 を指定することで発生中のエラー件数のみ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.

項目	型説明			
引数	VT_UI1			取得したいエラー数を指定します.
戻り値	VT_ARRAY | VT_VARIANT			
	0	VT_UI1		発生コントローラ内のエラー数を取得します.
	1	VT_ARRAY | VT_UI2		
		1.n	VT_UI2	エラーコード番号を指定したエラー件数分取得します. コントローラ内に指定した数のエラーがない場合は, 指定したエラー数分の 0 が入ります.

使用例

```
// エラーコードを2つ取得
object[] val = controller.Execute("GetError", 2) as object[];

int? errorNum = val[0] as int?; // エラー数を取得
int?[] error = val[1] as int?[]; //取得したエラーコード
```

2.2.2.4.2. ClearError コマンド

指定したエラーコードのシステムエラーを解除します. LJ-X8000A コントローラはすべてのエラーが解除された場合, 測定を開始します. 以下に引数と戻り値を示します.

項目	型説明	
引数	VT_UI2	解除したいエラーコードを指定します[2].
戻り値	特になし	

使用例

```
// エラーコード133のエラーを解除します.
controller.Execute("GetError", 133);
```

2.2.2.4.3. TrgErrorReset コマンド

TRG_ERROR の ON 状態をクリアします. TRG_ERROR は並列撮像が ON のとき, トリガー間隔が 10ms 以上空くと, ON 状態になります. TRG__ERROR 状態を確認する場合は, GetAttentionStatus で確認して下さい. 以下に引数と戻り値を示します.

項目	型説明
引数	特になし.
戻り値	特になし.

使用例

```
//TRG_ERRORON状態を解除します.
controller.Execute("TrgErrorReset");
```

2.2.2.4.4. GetTriggerAndPulseCount コマンド

トリガカウントの値とパルスカウントの値を取得します. 以下に引数と戻り値を示します.

項目	型説明
引数	特になし
戻り値	VT_ARRAY | VT_VARIANT

[2] エラーコードによっては削除できないものもあります.

項目	型説明		
	0	VT_UI4	トリガカウントの値を取得します. カウントの値範囲は, 0～4,294,967,295 で上限を超えると 0 に戻ります.
	1	VT_I4	パルスカウントの値を取得します. カウントの値範囲は-2,147,483,648～2,147,483,647 で正の上限を超えると負の上限の最大値となり, 負の上限を超えると正の上限の最大値になります.

使用例

```
// トリガーカウントパルスカウントを取得する
object[] val = controller.Execute("GetTriggerAndPulseCount", 2) as object[];

uint? triggerCount= val[0] as uint?; // トリガーカウントを取得
int? pulseCount = val[1] as int?;       //  パルスカウントを取得
```

2.2.2.4.5. GetHeadTemperature コマンド

ヘッド温度を読み出します. 以下に引数と戻り値を示します.

項目	型説明		
引数	特になし		
戻り値	VT_ARRAY | VT_I2		
	0	VT_I2	センサー(CMOS)温度を取得します.
	1	VT_I2	プロセッサー温度を取得します.
	2	VT_I2	ケース温度(筐体)を取得します. この項目は LJ-V ヘッドでは取得できないため(0xFFFF)の値が取得されます.

使用例

```
// ヘッドの温度を取得する
short?[] val = controller.Execute("GetHeadTempature") as short?[];
```

2.2.2.4.6. GetSerialNumber コマンド

コントローラとヘッドのシリアル番号を取得します. 以下に引数と戻り値を示します.

項目	型説明		
引数	特になし		
戻り値	VT_ARRAY | VT_BSTR		
	0	VT_BSTR	コントローラのシリアル番号を取得します.
	1	VT_BSTR	ヘッドのシリアル番号を取得します.

使用例

```
// シリアル番号を取得する
object[] val = controller.Execute("GetSerialNumber") as object[];

string controllerSerial = val[0] as string; // コントローラのシリアル番号を取得
string headSerial = val[1] as string;   //  ヘッダのシリアル番号を取得
```

2.2.2.4.7. GetAttentionStatus コマンド

TRG_ERROR/MEM_FULL/TRG_PASS の状態を取得します. 以下に引数と戻り値を示します.

項目	型説明		
引数	特になし		
戻り値	VT_ARRAY | VT_UI1		
	0	VT_UI1	TRG_ERROR 状態
	1	VT_UI1	MEM_FULL 状態
	2	VT_UI1	TRG_PASS 状態

使用例

```
// TRG_ERROR/MEM_FULL/TRG_PASS状態を取得します.
object[] val = controller.Execute("GetAttentionStatus") as object[];

byte? trgErrror = val[0] as byte?; // TRG_ERROR状態を取得
byte? memFull = val[1] as byte?; //  MEM_FULL状態を取得
byte? trgPass = val[2] as byte?; //  TRG_PASS状態を取得
```

2.2.2.4.8. GetLedLightImage コマンド

LED とレーザーの明るさを指定した内容で 1 画素 8bit の VGA(640×480)サイズの LED 照明画像を取得します. 本コマンドは, LASER_ON 端子が OFF の場合デバイスからエラーが返却されます. 以下に引数と戻り値を示します.

項目	型説明		
引数	VT_ARRAY | VT_UI1		
	0	VT_UI1	LED の明るさを指定します. 設定範囲:1～100
	1	VT_UI1	レーザーの明るさを指定します. 設定範囲:1～100
戻り値	VT_ARRAY | VT_UI1		
	n	VT_UI1	LED 照明画像のバイト配列を取得します.

使用例

```
// LED照明画像の取得
object[] val = controller.Execute("GetAttentionStatus") as object[];

byte?[] ledLightImage = val[0] as byte?[]; // LED照明画像のバイト列取得
```

2.2.2.4.9. Trigger コマンド

トリガーを発行します. コントローラがプロバイダからのトリガーコマンドを実行するためには, コントローラ側の設定で外部トリガー入力にしている必要があります. 以下に引数と戻り値を示します.

項目	型説明
引数	特になし.
戻り値	特になし.

使用例

```
//トリガーを発行する
controller.Execute("Trigger");
```

2.2.2.4.10. StartMeasure コマンド

バッチ測定を開始します. 本コマンドは, バッチ測定が OFF または複数コントローラ同期機能を使用しているが, 指定したデバイスが「同期マスター」に設定されていない場合, LASER_ON 端子が OFF の場合はデバイスからエラーが返却されます. 以下に引数と戻り値を示します.

項目	型説明
引数	特になし.
戻り値	特になし.

使用例

```
// バッチ測定を開始する
controller.Execute("StartMeasure");
```

2.2.2.4.11. StopMeasure コマンド

バッチ測定を終了します. バッチ測定が OFF または複数コントローラ同期機能を使用しているが, 指定したデバイスが「同期マスター」に設定されていない場合, LASER_ON 端子が OFF の場合はデバイスからエラーが返却されます. 以下に引数と戻り値を示します.

項目	型説明
引数	特になし.
戻り値	特になし.

使用例

```
//バッチ測定を終了する
controller.Execute("StopMeasure");
```

2.2.2.4.12. ClearMemory コマンド

コントローラ内に蓄積されたプロファイルデータをクリアします. 以下に引数と戻り値を示します.

項目	型説明
引数	特になし.
戻り値	特になし.

使用例

```
//コントローラ内に蓄積されているデータをクリアする
controller.Execute("ClearMemory");
```

2.2.2.4.13. GetActiveProgram コマンド

現在のアクティブプログラム No. を取得します. 以下に引数と戻り値を示します.

項目	型説明	
引数	特になし.	
戻り値	VT_UI1	アクティブプログラム No. を取得します.

使用例

```
//現在のアクティブプログラムNo.を取得します.
byte? activeProgram = controller.Execute("GetActiveProgram") as byte?;
```

2.2.2.4.14. ChangeActiveProgram コマンド

アクティブプログラム No. を切り替えます. 以下に引数と戻り値を示します.

項目	型説明	
引数	VT_UI1	切り替え後のアクティブプログラム No. を指定します.
戻り値	特になし.	

使用例

```
//アクティブプログラム番号を5に切り替えます.
byte activeProgramNo = 5;
controller.Execute("ChangeActiveProgram", activeProgramNo);
```

2.2.2.4.15. GetProfile コマンド

プロファイルデータを取得します. 以下に引数と戻り値を示します. 引数指定値の各種項目に関し

ては,「LJ-X8000A 通信ライブラリ リファレンスマニュアル」を参照してください.

<table>
<tr><th>項目</th><th colspan="5">型説明</th></tr>
<tr><td rowspan="6">引数</td><td colspan="5">VT_ARRAY | VT_VARIANT</td></tr>
<tr><td>0</td><td>VT_UI1</td><td colspan="3">アクティブ面/非アクティブ面のどちらから取得するかを指定します.
<table><tr><th>設定値</th><th>概要</th></tr><tr><td>0</td><td>アクティブ面</td></tr><tr><td>1</td><td>非アクティブ面</td></tr></table></td></tr>
<tr><td>1</td><td>VT_UI1</td><td colspan="3">プロファイル取得位置の指定方法を指定します.
<table><tr><th>設定値</th><th>概要</th></tr><tr><td>0</td><td>最新から</td></tr><tr><td>1</td><td>最古から</td></tr><tr><td>2</td><td>任意の位置指定</td></tr></table></td></tr>
<tr><td>2</td><td>VT_UI4</td><td colspan="3">プロファイル位置指定方法で 2 を指定した場合, 取得対象のプロファイル No. を指定します.
値範囲：0~4294967295
※指定したプロファイル番号が存在しない場合はエラーが返却されます.</td></tr>
<tr><td>3</td><td>VT_UI1</td><td colspan="3">読み出すプロファイル数を指定します.
値範囲：1~255</td></tr>
<tr><td>4</td><td>VT_UI1</td><td colspan="3">読み出したプロファイルとそれ以前のプロファイルを消去するか指定します.
<table><tr><th>設定値</th><th>概要</th></tr><tr><td>0</td><td>消去しない</td></tr><tr><td>1</td><td>消去する</td></tr></table></td></tr>
<tr><td rowspan="10">戻り値</td><td colspan="5">VT_ARRAY | VT_VARIANT</td></tr>
<tr><td>0</td><td colspan="4">VT_ARRAY | VT_VARIANT</td></tr>
<tr><td></td><td>0.0</td><td>VT_UI1</td><td colspan="2">1 単位のデータに格納されるプロファイル数を取得します.</td></tr>
<tr><td></td><td>0.1</td><td>VT_UI1</td><td colspan="2">輝度の ON/OFF を取得します.</td></tr>
<tr><td></td><td>0.2</td><td>VT_UI2</td><td colspan="2">1 プロファイル中のデータ点数を取得します.</td></tr>
<tr><td></td><td>0.3</td><td>VT_I4</td><td colspan="2">1 点目の X 座標を取得します.</td></tr>
<tr><td></td><td>0.4</td><td>VT_I4</td><td colspan="2">データ点の X 方向の間隔</td></tr>
<tr><td></td><td>0.5</td><td>VT_I4</td><td colspan="2">取得時点の最新プロファイル No. を取得します.</td></tr>
<tr><td></td><td>0.6</td><td>VT_UI4</td><td colspan="2">コントローラが保持する最古のプロファイル No. を取得します.</td></tr>
<tr><td></td><td>0.7</td><td>VT_UI4</td><td colspan="2">読み出した中で最古のプロファイル No. を取得します.</td></tr>
</table>

項目	型説明			
		0.8	VT_UI1	今回読み出したプロファイル数を取得します.
	1.n	VT_ARRAY|VT_I4		読み出したプロファイルデータを取得します. 取得したプロファイルデータが順に格納されます.

使用例

```
//10番目のプロファイルデータから1個のプロファイルデータを取得する
object[] prm = new object[]{0,2,10,1,0 };
object[] retValue = controller.Execute("GetProfile",prm) as object[];

object[] profileInfo = retValue[0] as object[]; // プロファイル情報

byte? ProfileNum = profileInfo [0] as byte?; // プロファイル数
byte? Luminication = profileInfo [1] as byte?; // 輝度情報
short? ProfileDataCount profileInfo [2] as short?; // プロファイルデータ点数
int? XPos = profileInfo [3] as int?; // X座標
int? XInterval = profileInfo [4] as int?; // X方向の間隔
uint? CurrentProfileNo = profileInfo [5] as uint?; // 最新のプロファイル番号
uint? OldestProfileNo = profileInfo [6] as uint?; // 最古のプロファイル番号
byte? GetProfileCount = profileInfo [7] as byte?; // 今回取得したプロファイル数

int?[] profileData = retValue[1] as int[]?; // プロファイルデータ
```

2.2.2.4.16. GetBatchProfile コマンド

バッチプロファイルデータを取得します. 以下に引数と戻り値を示します.

項目	型説明		
引数	VT_ARRAY | VT_VARIANT		
	0	VT_UI1	アクティブ面/非アクティブ面のどちらから取得するかを指定します.
			設定値: 0 — 概要: アクティブ面
			設定値: 1 — 概要: 非アクティブ面
	1	VT_UI1	バッチ取得位置の指定方法を指定します.
			設定値: 0 — 概要: 最新から
			設定値: 2 — 概要: 任意の位置指定
			設定値: 3 — 概要: バッチ確定後最新から
			設定値: 4 — 概要: 最新のみ

<table>
<tr><th>項目</th><th colspan="5">型説明</th></tr>
<tr><td></td><td>2</td><td>VT_UI4</td><td colspan="3">バッチ取得位置指定方法で 2 を指定した場合，取得対象のバッチ No. を指定します.
値範囲：0~4294967295
※指定したバッチ No. が存在しない場合はエラーが返却されます.</td></tr>
<tr><td></td><td>3</td><td>VT_UI4</td><td colspan="3">バッチ内の取得開始プロファイル No. を指定します.
値範囲：0~4294967295
※指定したプロファイル No. が存在しない場合はエラーが返却されます.</td></tr>
<tr><td></td><td>4</td><td>VT_UI1</td><td colspan="3">読み出すプロファイル数を指定します.
値範囲：1~255</td></tr>
<tr><td></td><td>5</td><td>VT_UI1</td><td colspan="3">読み出したプロライフとそれ以前のプロファイルを消去するか指定します.
<table><tr><th>設定値</th><th>概要</th></tr><tr><td>0</td><td>消去しない</td></tr><tr><td>1</td><td>消去する</td></tr></table></td></tr>
<tr><td rowspan="16">戻り値</td><td colspan="5">VT_ARRAY | VT_VARIANT</td></tr>
<tr><td>0</td><td colspan="4">VT_ARRAY | VT_VARIANT</td></tr>
<tr><td></td><td>0.0</td><td>VT_UI1</td><td colspan="2">1 単位のデータに格納されるプロファイル数を取得します.</td></tr>
<tr><td></td><td>0.1</td><td>VT_UI1</td><td colspan="2">輝度の ON/OFF を取得します.</td></tr>
<tr><td></td><td>0.2</td><td>VT_UI2</td><td colspan="2">1 プロファイル中のデータ点数を取得します.</td></tr>
<tr><td></td><td>0.3</td><td>VT_I4</td><td colspan="2">1 点目の X 座標を取得します.</td></tr>
<tr><td></td><td>0.4</td><td>VT_I4</td><td colspan="2">データ点の X 方向の間隔</td></tr>
<tr><td></td><td>0.5</td><td>VT_I4</td><td colspan="2">取得時点の最新プロファイル No. を取得します.</td></tr>
<tr><td></td><td>0.6</td><td>VT_UI4</td><td colspan="2">コントローラが保持する最古のプロファイル No. を取得します.</td></tr>
<tr><td></td><td>0.7</td><td>VT_UI4</td><td colspan="2">コントローラが保持する最古のバッチ内のプロファイル数を取得します.</td></tr>
<tr><td></td><td>0.8</td><td>VT_UI4</td><td colspan="2">読み出した中で最古のプロファイル No. を取得します.</td></tr>
<tr><td></td><td>0.9</td><td>VT_UI4</td><td colspan="2">今回読み出したバッチ No. を取得します.</td></tr>
<tr><td></td><td>0.10</td><td>VT_UI4</td><td colspan="2">今回読み出したバッチ内のプロファイル数を取得します.</td></tr>
<tr><td></td><td>0.11</td><td>VT_UI4</td><td colspan="2">読み出したバッチ内の何番目のプロファイルか取得します.</td></tr>
<tr><td></td><td>0.12</td><td>VT_UI1</td><td colspan="2">今回読み出したプロファイル数を取得します.</td></tr>
<tr><td></td><td>0.13</td><td>VT_UI1</td><td colspan="2">最新バッチ測定が完了しているか取得します.</td></tr>
</table>

項目	型説明		
	1.n	VT_ARRAY|VT_I4	読み出したプロファイルデータを取得します. 取得したプロファイルデータが順に格納されます.

使用例

```
//バッチ1番目から10番目のプロファイルデータから1個のプロファイルデータを取得する
object[] prm = new object[]{0,2,1,10,1,0 };
object[] retValue = controller.Execute("GetBatchProfile",prm) as object[];

object[] profileInfo = retValue[0] as object[]; // プロファイル情報

byte? ProfileNum = profileInfo [0] as byte?; // プロファイル数
byte? Luminication = profileInfo [1] as byte?; // 輝度情報
short? ProfileDataCount profileInfo [2] as short?; // プロファイルデータ点数
int? XPos = profileInfo [3] as int?; // X座標
int? XInterval = profileInfo [4] as int?; // X方向の間隔
uint? CurrentProfileNo = profileInfo [5] as uint?; // 最新のプロファイル番号
uint? OldestProfileNo = profileInfo [6] as uint?; // 最古のプロファイル番号
uint? GetProfileCount = profileInfo [7] as uint?; // 最古のバッチ内のプロファイル数
uint? GetOldestProfNo = profileInfo [8] as uint?; //読み出した中で最古のプロファイルNo.
uint? GetBatchNo = profileInfo [9] as uint?; //読み出したバッチNo.
uint? GetProfileNo = profileInfo [10] as uint?; //読み出したバッチ内のプロファイル数
uint? GetBatchProfileNo = profileInfo [11] as uint?; //何番目のプロファイルか
byte? GetProfileCount = profileInfo [12] as uint?; //読み出した中で最古のプロファイルNo.
byte? isBatchMesure = profileInfo [13] as byte?; //バッチ測定が終了しているか

int?[] profileData = retValue[1] as int[]?; // プロファイルデータ
```

2.2.2.4.17. GetBatchSimpleArray コマンド

バッチプロファイルデータを SimpleArray で取得します. 以下に引数と戻り値を示します.

<table>
<tr><th>項目</th><th colspan="4">型説明</th></tr>
<tr><td rowspan="8">引数</td><td colspan="4">VT_ARRAY | VT_VARIANT</td></tr>
<tr><td>0</td><td>VT_UI1</td><td colspan="2">アクティブ面/非アクティブ面のどちらから取得するかを指定します.
<table><tr><th>設定値</th><th>概要</th></tr><tr><td>0</td><td>アクティブ面</td></tr><tr><td>1</td><td>非アクティブ面</td></tr></table></td></tr>
<tr><td>1</td><td>VT_UI1</td><td colspan="2">バッチ取得位置の指定方法を指定します.
<table><tr><th>設定値</th><th>概要</th></tr><tr><td>0</td><td>最新から</td></tr><tr><td>2</td><td>任意の位置指定</td></tr><tr><td>3</td><td>バッチ確定後最新から</td></tr><tr><td>4</td><td>最新のみ</td></tr></table></td></tr>
<tr><td>2</td><td>VT_UI4</td><td colspan="2">バッチ取得位置指定方法で 2 を指定した場合, 取得対象のバッチ No.を指定します.
値範囲：0~4294967295
※指定したバッチ No.が存在しない場合はエラーが返却されます.</td></tr>
<tr><td>3</td><td>VT_UI4</td><td colspan="2">バッチ内の取得開始プロファイル No.を指定します.
値範囲：0~4294967295
※指定したプロファイル No.が存在しない場合はエラーが返却されます.</td></tr>
<tr><td>4</td><td>VT_UI1</td><td colspan="2">読み出すプロファイル数を指定します.
値範囲：1~255</td></tr>
<tr><td>5</td><td>VT_UI1</td><td colspan="2">読み出したプロライフとそれ以前のプロファイルを消去するか指定します.
<table><tr><th>設定値</th><th>概要</th></tr><tr><td>0</td><td>消去しない</td></tr><tr><td>1</td><td>消去する</td></tr></table></td></tr>
<tr><td rowspan="6">戻り値</td><td colspan="4">VT_ARRAY | VT_VARIANT</td></tr>
<tr><td>0</td><td colspan="3">VT_ARRAY | VT_VARIANT</td></tr>
<tr><td></td><td>0.0</td><td>VT_UI1</td><td>1 単位のデータに格納されるプロファイル数を取得します.</td></tr>
<tr><td></td><td>0.1</td><td>VT_UI1</td><td>輝度の ON/OFF を取得します.</td></tr>
<tr><td></td><td>0.2</td><td>VT_UI2</td><td>1 プロファイル中のデータ点数を取得します.</td></tr>
<tr><td></td><td>0.3</td><td>VT_I4</td><td>1 点目の X 座標を取得します.</td></tr>
</table>

項目	型説明			
		0.4	VT_I4	データ点の X 方向の間隔
		0.5	VT_I4	取得時点の最新プロファイル No. を取得します.
		0.6	VT_UI4	コントローラが保持する最古のプロファイル No. を取得します.
		0.7	VT_UI4	コントローラが保持する最古のバッチ内のプロファイル数を取得します.
		0.8	VT_UI4	読み出した中で最古のプロファイル No. を取得します.
		0.9	VT_UI1	今回読み出したバッチ No. を取得します.
		0.10	VT_UI4	今回読み出したバッチ内のプロファイル数を取得します.
		0.11	VT_UI4	読み出した中で最古のプロファイルが, バッチ内の何番目のプロファイルか取得します.
		0.12	VT_UI1	今回読み出したプロファイル数を取得します.
		0.13	VT_UI1	最新バッチ測定が完了しているか取得します.
	1.n	VT_ARRAY|VT_I4		読み出したプロファイルデータのヘッダ情報が取得されます.
	2.n	VT_ARRAY|VT_UI2		読み出したプロファイルデータの高さ情報が取得されます.
	3.n	VT_ARRAY|VT_UI2		読み出したプロファイルデータの輝度情報が取得されます.

使用例

```
//10番目のプロファイルデータから1個のプロファイルデータを取得する
object[] prm = new object[]{0,2,10,1,0 };
object[] retValue = controller.Execute("GetBatchProfileSimpleArray",prm) as object[];

object[] profileInfo = retValue[0] as object[]; // プロファイル情報

byte? ProfileNum = profileInfo [0] as byte?; // プロファイル数
byte? Luminication = profileInfo [1] as byte?; // 輝度情報
ushort? ProfileDataCount profileInfo [2] as ushort?; // プロファイルデータ点数
int? XPos = profileInfo [3] as int?; // X座標
int? XInterval = profileInfo [4] as int?; // X方向の間隔
uint? CurrentProfileNo = profileInfo [5] as uint?; // 最新のプロファイル番号
uint? OldestProfileNo = profileInfo [6] as uint?; // 最古のプロファイル番号
uint? GetProfileCount = profileInfo [7] as uint?; // 最古のバッチ内のプロファイル数
uint? GetOldestProfNo = profileInfo [8] as uint?; //読み出した中で最古のプロファイルNo.
uint? GetBatchNo = profileInfo [9] as uint?; //読み出したバッチNo.
uint? GetProfileNo = profileInfo [10] as uint?; //読み出したバッチ内のプロファイル数
uint? GetBatchProfileNo = profileInfo [11] as uint?; //何番目のプロファイルか
byte? GetProfileCount = profileInfo [12] as uint?; //読み出した中で最古のプロファイルNo.
byte? isBatchMesure = profileInfo [13] as byte?; //バッチ測定が終了しているか
```

```
int?[] profileData = retValue[1] as int[]?; // プロファイルヘッダーデータ
ushort?[] profileData = retValue[2] as ushort[]?; // プロファイル高さデータ
ushort?[] profileData = retValue[3] as ushort[]?; // プロファイル輝度データ
```

2.2.2.4.18. SetProfileCount コマンド

プロファイルデータを取得する際に API に渡すバッファサイズを指定します.

本プロバイダは，ライブラリの API を使いプロファイルデータを取得します. API 実行の際にあらかじめ取得するプロファイルデータ数分のメモリを確保する必要があり，プロバイダ側では初期状態でメモリを最大バッファ(1 プロファイルにつき 25628 バイト)確保しています. 内訳は高さデータ(3200 点×4 バイト)，輝度データ（3200×4 バイト)，プロファイルデータヘッダ(24 バイト)，プロファイルデータフッダ(4 バイト)となっています. メモリ不足等で取得ができない等ありましたら本コマンドでバッファサイズを小さくしてください. バッファサイズを小さくした場合は，コントローラ側でもそのサイズになるように設定をする必要があります.

※コントローラ側の設定で使用ヘッドが LJX シリーズ測定範囲=1/2,間引き X 軸=FULL, サンプリング周期=16kHz の場合は,3200(LJX ヘッド基準プロファイル点数)×0.5×1.0×0.5=800 となります. コマンドの引数として(2,0,1)を設定すると同じ値になります.

<table>
<tr><th>項目</th><th colspan="3">型説明</th></tr>
<tr><td rowspan="3">引数</td><td colspan="3">VT_ARRAY | VT_UI1</td></tr>
<tr><td>0</td><td>VT_UI1</td><td>測定範囲 X 方向を指定します.
<table><tr><th>指定値</th><th>コントローラ側設定値</th><th>係数</th></tr><tr><td>0</td><td>OFF</td><td>1.0</td></tr><tr><td>1</td><td>3/4</td><td>0.75</td></tr><tr><td>2</td><td>1/2</td><td>0.5</td></tr><tr><td>3</td><td>1/4</td><td>0.25</td></tr></table></td></tr>
<tr><td>1</td><td>VT_UI1</td><td>間引き X 軸を指定します.
<table><tr><th>指定値</th><th>コントローラ側設定値</th><th>係数</th></tr><tr><td>0</td><td>OFF</td><td>1.0</td></tr><tr><td>1</td><td>1/2</td><td>0.5</td></tr><tr><td>2</td><td>1/4</td><td>0.25</td></tr></table></td></tr>
</table>

<table>
<tr><th>項目</th><th colspan="3">型説明</th></tr>
<tr><td></td><td>2</td><td>VT_UI1</td><td>LJ-X シリーズのヘッドと LJ-V シリーズのヘッドで指定する項目が異なります.

LJ-X ヘッド：サンプリング周期
<table>
<tr><th>指定値</th><th>コントローラ側設定値</th><th>係数</th></tr>
<tr><td>0</td><td>8kHz または 16kHz 以外の設定</td><td>1.0</td></tr>
<tr><td>1</td><td>8kHz, 16kHz</td><td>0.5</td></tr>
</table>
LJ-V ヘッド：ビニング
<table>
<tr><th>指定値</th><th>コントローラ側設定値</th><th>係数</th></tr>
<tr><td>0</td><td>OFF</td><td>1.0</td></tr>
<tr><td>1</td><td>ON</td><td>0.5</td></tr>
</table></td></tr>
<tr><td>戻り値</td><td colspan="3">特になし.</td></tr>
</table>

使用例

```
// LJXヘッド時にデータ点数を800点にする
//(高さ(800×4バイト)+輝度(800×4バイト)+ヘッダ(24)+フッダ(4)=6428バイト
object[] prm = new object[]{2,1,0 };
controller.Execute("SetProfileCount ", prm);
```

2.2.2.4.19. StartHighSpeedDataCommunication コマンド

高速データ通信を開始します. 以下に引数と戻り値を示します.

<table>
<tr><th>項目</th><th colspan="3">型説明</th></tr>
<tr><td rowspan="3">引数</td><td colspan="3">VT_ARRAY | VT_VARIANT</td></tr>
<tr><td>0</td><td>VT_UI2</td><td>高速データに用いるポート番号</td></tr>
<tr><td>1</td><td>VT_UI4</td><td>まとめて送信するプロファイル数
指定した数のプロファイルを受信すると OnMessage イベントが発生します.</td></tr>
<tr><td>戻り値</td><td colspan="3">特になし.</td></tr>
</table>

使用例

```
// 高速データ通信を開始する.
object[] prm = new object[]{28461, 10};
controller.Execute("StartHighSpeedDataCommunication", prm);
```

2.2.2.4.20. StartHighSpeedDataCommunicationSimpleArray コマンド

高速データ通信を SimpleArray モードで開始します. 以下に引数と戻り値を示します.

項目	型	説明	
引数	VT_ARRAY | VT_VARIANT		
	0	VT_UI2	高速データに用いるポート番号
	1	VT_UI4	まとめて送信するプロファイル数 指定した数のプロファイルを受信すると OnMessage イベントが発生します.
戻り値	特になし.		

使用例

```
// 高速データ通信をSimpleArrayモードで開始する.
object[] prm = new object[]{28461, 10};
controller.Execute("StartHighSpeedDataCommunicationSimpleArray", prm);
```

2.2.2.4.21. StopHighSpeedDataCommunication コマンド

高速データ通信を終了します. 以下に引数と戻り値を示します.

項目	型説明
引数	特になし.
戻り値	特になし.

使用例

```
// 高速データ通信を終了します.
controller.Execute("StopHighSpeedDataCommunication ");
```

2.2.2.5. 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. 受け取ることが可能なイベントについては 2.4 を参照してください.

2.2.3. CaoVariable クラス

2.2.3.1. Value プロパティ

変数名によって動作が異なります. 詳細は, 2.3.変数一覧を参照してください.

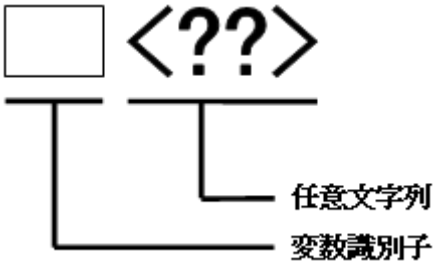
2.3. 変数一覧

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．

なお変数は，CaoVariable クラスのオブジェクトを指します．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



2.3.1. 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.28
@VERSION	DLL バージョンを取得します.	○	－	P.28
CURRENT_PROFILE<??>	最新からプロファイルデータを取得します.	○	－	P.29
OLDEST_PROFILE<??>	最古からプロファイルデータを取得します.	○	－	P.30
SPEC_PROFILE<??>	位置指定してプロファイルデータを取得します.	○	－	P.31
CURRENT_BATCHPROFILE<??>	最新のバッチからプロファイルデータを取得する.	○	－	P.32
SPEC_BATCHPROFILE<??>	位置指定をして特定のバッチからプロファイルデータを取得します.	○	－	P.34
COMMITED_BATCHPROFILE<??>	バッチ確定後最新のバッチからプロファイルデータを取得します.	○	－	P.35
CURRENTONLY_BATCHPROFILE<??>	最新バッチから最新のみプロファイルデータのみ取得します.	○	－	P.36

CURRENT_BATCHPROFILE_SIMPLE<??>	最新のバッチからプロファイルデータを取得する.	○	-	P. 38
SPEC_BATCHPROFILE_SIMPLE<??>	位置指定をして特定のバッチからプロファイルデータを取得します.	○	-	P. 39
COMMITED_BATCHPROFILE_SIMPLE<??>	バッチ確定後最新のバッチからプロファイルデータを取得します.	○	-	P. 41
CURRENTONLY_BATCHPROFILE_SIMPLE<??>	最新バッチから最新のみプロファイルデータのみ取得します.	○	-	P. 42

2.3.1.1. @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

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

2.3.1.2. @VERSION

DLL のバージョンの取得をします.

データ型

型説明	
VT_BSTR	DLL のバージョンを取得します. *.*.*

使用例

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

2.3.1.3. CURRENT_PROFILE<??>

最新からプロファイルデータを指定数取得します.オプションで取得するプロファイルデータの数を指定し指定した数のプロファイルデータを取得します.

取得されるデータは以下に示します.

型説明			
VT_ARRAY | VT_VARIANT			
0	VT_ARRAY | VT_VARIANT		
	0.0	VT_UI1	1 単位のデータに格納されるプロファイル数を取得します.
	0.1	VT_UI1	輝度の ON/OFF を取得します.
	0.2	VT_UI2	1 プロファイル中のデータ点数を取得します.
	0.3	VT_I4	1 点目の X 座標を取得します.
	0.4	VT_I4	データ点の X 方向の間隔
	0.5	VT_I4	取得時点の最新プロファイル No.を取得します.
	0.6	VT_UI4	コントローラが保持する最古のプロファイル No.を取得します.
	0.7	VT_UI4	読み出した中で最古のプロファイル No.を取得します.
	0.8	VT_UI1	今回読み出したプロファイル数を取得します.
1.n	VT_ARRAY|VT_I4		読み出したプロファイルデータを取得します. 取得したプロファイルデータが順に格納されます.

オプション	必須	説明	値範囲
TargetBank=[=<取得面>]	--	アクティブ面/非アクティブ面のどちらから取得するかを指定します. 設定値 / 概要 0 / アクティブ面 1 / 非アクティブ面 デフォルト:0	0-1
ProfileCount [=<取得するプロファイル数>]	○	読み出すプロファイル数を指定します.	1-255
Erase=[<プロファイルデータ削除有無>]	--	読み出したプロファイルとそれ以前のプロファイルを消去するか指定します. 設定値 / 概要 0 / 消去しない 1 / 消去する	0-1

使用例

```
// 取得するプロファイル数は１つで変数を追加
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("CURRENT_PROFILE01","ProfileCou
nt=1");

//プロファイルデータを取得する
object[] retValue = var.Value as object[];

object[] profileInfo = retValue[0] as object[]; // プロファイル情報

byte? ProfileNum = profileInfo [0] as byte?; // プロファイル数
byte? Luminication = profileInfo [1] as byte?; // 輝度情報
short? ProfileDataCount profileInfo [2] as short?; // プロファイルデータ点数
int? XPos = profileInfo [3] as int?; // X座標
int? XInterval = profileInfo [4] as int?; // X方向の間隔
uint? CurrentProfileNo = profileInfo [5] as uint?; // 最新のプロファイル番号
uint? OldestProfileNo = profileInfo [6] as uint?; // 最古のプロファイル番号
byte? GetProfileCount = profileInfo [7] as byte?; // 今回取得したプロファイル数

int?[] profileData = retValue[1] as int[]?; // プロファイルデータ
```

2.3.1.4. OLDEST_PROFILE<??>

最古からプロファイルデータを取得します.オプションで取得するプロファイルデータの数を指定し指定した数のプロファイルデータを取得します.

取得されるデータは 2.3.1.3 と同じになっておりますのでそちらを参照ください.

<table>
<tr><th>オプション</th><th>必須</th><th>説明</th><th>値範囲</th></tr>
<tr><td>TargetBank=[=<取得面>]</td><td>--</td><td>アクティブ面/非アクティブ面のどちらから取得するかを指定します.
<table><tr><th>設定値</th><th>概要</th></tr><tr><td>0</td><td>アクティブ面</td></tr><tr><td>1</td><td>非アクティブ面</td></tr></table>デフォルト:0</td><td>0-1</td></tr>
<tr><td>ProfileCount
[=<取得するプロファイル数>]</td><td>○</td><td>読み出すプロファイル数を指定します.</td><td>1-255</td></tr>
<tr><td>Erase=[<プロファイルデータ削除有無>]</td><td>--</td><td>読み出したプロファイルとそれ以前のプロファイルを消去するか指定します.
<table><tr><th>設定値</th><th>概要</th></tr><tr><td>0</td><td>消去しない</td></tr><tr><td>1</td><td>消去する</td></tr></table></td><td>0-1</td></tr>
</table>

使用例

```
// 取得するプロファイル数は１つで変数を追加
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("OLDEST_PROFILE01","ProfileCount=1");

//プロファイルデータを取得する
object[] retValue = var.Value as object[];

object[] profileInfo = retValue[0] as object[]; // プロファイル情報

byte? ProfileNum = profileInfo [0] as byte?; // プロファイル数
byte? Luminication = profileInfo [1] as byte?; // 輝度情報
short? ProfileDataCount profileInfo [2] as short?; // プロファイルデータ点数
int? XPos = profileInfo [3] as int?; // X座標
int? XInterval = profileInfo [4] as int?; // X方向の間隔
uint? CurrentProfileNo = profileInfo [5] as uint?; // 最新のプロファイル番号
uint? OldestProfileNo = profileInfo [6] as uint?; // 最古のプロファイル番号
byte? GetProfileCount = profileInfo [7] as byte?; // 今回取得したプロファイル数

int?[] profileData = retValue[1] as int[]?; // プロファイルデータ
```

2.3.1.5. SPEC_PROFILE<??>

指定位置した場所からプロファイルデータ取得します. オプションで取得するプロファイル No. を指定しその場所から指定した数のプロファイルデータを取得します.

取得されるデータは 2.3.1.3 と同じになっておりますのでそちらを参照ください.

<table>
<tr><th>オプション</th><th>必須</th><th>説明</th><th>値範囲</th></tr>
<tr><td>TargetBank=[=<取得面>]</td><td>--</td><td>アクティブ面/非アクティブ面のどちらから取得するかを指定します.
<table><tr><th>設定値</th><th>概要</th></tr><tr><td>0</td><td>アクティブ面</td></tr><tr><td>1</td><td>非アクティブ面</td></tr></table>デフォルト:0</td><td>0-1</td></tr>
<tr><td>ProfileNo=[=<取得開始するプロファイル No>]</td><td>○</td><td>プロファイル位置指定方法で 2 を指定した場合, 取得対象のプロファイル No. を指定します.</td><td>0~4294967295</td></tr>
<tr><td>ProfileCount [=<取得するプロファイル数>]</td><td>○</td><td>読み出すプロファイル数を指定します.</td><td>1-255</td></tr>
<tr><td>Erase=[<プロファイルデータ削除有無>]</td><td>--</td><td>読み出したプロファイルとそれ以前のプロファイルを消去するか指定します.</td><td>0-1</td></tr>
</table>

オプション	必須	説明		値範囲
		設定値	概要	
		0	消去しない	
		1	消去する	

使用例

```
// プロファイルNo１０からプロファイル数を１つ取得する変数を追加
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("SPEC_PROFILE01","ProfileCount=
1,ProfileNo=10");

//プロファイルデータを取得する
object[] retValue = var.Value as object[];

object[] profileInfo = retValue[0] as object[]; // プロファイル情報

byte? ProfileNum = profileInfo [0] as byte?; // プロファイル数
byte? Luminication = profileInfo [1] as byte?; // 輝度情報
short? ProfileDataCount profileInfo [2] as short?; // プロファイルデータ点数
int? XPos = profileInfo [3] as int?; // X座標
int? XInterval = profileInfo [4] as int?; // X方向の間隔
uint? CurrentProfileNo = profileInfo [5] as uint?; // 最新のプロファイル番号
uint? OldestProfileNo = profileInfo [6] as uint?; // 最古のプロファイル番号
byte? GetProfileCount = profileInfo [7] as byte?; // 今回取得したプロファイル数

int?[] profileData = retValue[1] as int[]?; // プロファイルデータ
```

2.3.1.6. CURRENT_BATCHPROFILE<??>

最新からプロファイルデータを取得します.オプションで取得するプロファイルデータの数を指定し指定した数のプロファイルデータを取得します.

取得されるデータを以下に示します.

型説明			
VT_ARRAY | VT_VARIANT			
0	VT_ARRAY | VT_VARIANT		
	0.0	VT_UI1	1 単位のデータに格納されるプロファイル数を取得します.
	0.1	VT_UI1	輝度の ON/OFF を取得します.
	0.2	VT_UI2	1 プロファイル中のデータ点数を取得します.
	0.3	VT_I4	1 点目の X 座標を取得します.
	0.4	VT_I4	データ点の X 方向の間隔
	0.5	VT_I4	取得時点の最新プロファイル No. を取得します.

	0.6	VT_UI4	コントローラが保持する最古のプロファイル No. を取得します.
	0.7	VT_UI4	コントローラが保持する最古のバッチ内のプロファイル数を取得します.
	0.8	VT_UI4	読み出した中で最古のプロファイル No. を取得します.
	0.9	VT_UI4	今回読み出したバッチ No. を取得します.
	0.10	VT_UI4	今回読み出したバッチ内のプロファイル数を取得します.
	0.11	VT_UI4	読み出したバッチ内の何番目のプロファイルか取得します.
	0.12	VT_UI1	今回読み出したプロファイル数を取得します.
	0.13	VT_UI1	最新バッチ測定が完了しているか取得します.
1.n	VT_ARRAY|VT_I4		読み出したプロファイルデータを取得します. 取得したプロファイルデータが順に格納されます.

オプション	必須	説明	値範囲
TargetBank=[=<取得面>]	--	アクティブ面/非アクティブ面のどちらから取得するかを指定します. 設定値 / 概要 0 / アクティブ面 1 / 非アクティブ面 デフォルト:0	0-1
ProfileCount [=<取得するプロファイル数>]	○	読み出すプロファイル数を指定します.	1-255
Erase=[<プロファイルデータ削除有無>]	--	読み出したプロファイルとそれ以前のプロファイルを消去するか指定します. 設定値 / 概要 0 / 消去しない 1 / 消去する	0-1

使用例

```
//プロファイルデータを１つ取得する変数を追加
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("CURRENT_BATCHPROFILE01","Profi
leCount=1");

//プロファイルデータを取得する
object[] retValue = var.Value as object[];


object[] profileInfo = retValue[0] as object[]; // プロファイル情報
```

```
byte? ProfileNum = profileInfo [0] as byte?; // プロファイル数
byte? Luminication = profileInfo [1] as byte?; // 輝度情報
short? ProfileDataCount profileInfo [2] as short?; // プロファイルデータ点数
int? XPos = profileInfo [3] as int?; // X座標
int? XInterval = profileInfo [4] as int?; // X方向の間隔
uint? CurrentProfileNo = profileInfo [5] as uint?; // 最新のプロファイル番号
uint? OldestProfileNo = profileInfo [6] as uint?; // 最古のプロファイル番号
uint? GetProfileCount = profileInfo [7] as uint?; // 最古のバッチ内のプロファイル数
uint? GetOldestProfNo = profileInfo [8] as uint?; //読み出した中で最古のプロファイルNo.
uint? GetBatchNo = profileInfo [9] as uint?; //読み出したバッチNo.
uint? GetProfileNo = profileInfo [10] as uint?; //読み出したバッチ内のプロファイル数
uint? GetBatchProfileNo = profileInfo [11] as uint?; //何番目のプロファイルか
byte? GetProfileCount = profileInfo [12] as uint?; //読み出した中で最古のプロファイルNo.
byte? isBatchMesure = profileInfo [13] as byte?; //バッチ測定が終了しているか

int?[] profileData = retValue[1] as int[]?; // プロファイルデータ
```

2.3.1.7. SPEC_BATCHPROFILE<??>

位置指定した場所からプロファイルデータを取得します．オプションで取得するバッチ No とプロファイル No. を指定しその場所から指定した数のプロファイルデータを取得します．

取得されるデータは 2.3.1.6 と同じになっておりますのでそちらを参照ください．

<table>
<tr><th>オプション</th><th>必須</th><th>説明</th><th>値範囲</th></tr>
<tr><td>TargetBank=[=<取得面>]</td><td>--</td><td>アクティブ面/非アクティブ面のどちらから取得するかを指定します．
<table><tr><th>設定値</th><th>概要</th></tr><tr><td>0</td><td>アクティブ面</td></tr><tr><td>1</td><td>非アクティブ面</td></tr></table>デフォルト:0</td><td>0-1</td></tr>
<tr><td>BatchNo=[=<バッチ No>]</td><td>○</td><td>プロファイルデータを取得するバッチ No を指定します．</td><td>0~4294967295</td></tr>
<tr><td>ProfileNo=[=<取得開始するプロファイル No>]</td><td>○</td><td>プロファイル位置指定方法で 2 を指定した場合，取得対象のプロファイル No. を指定します．</td><td>0~4294967295</td></tr>
<tr><td>ProfileCount [=<取得するプロファイル数>]</td><td>○</td><td>読み出すプロファイル数を指定します．</td><td>1-255</td></tr>
<tr><td>Erase=[<プロファイルデータ削除有無>]</td><td>--</td><td>読み出したプロファイルとそれ以前のプロファイルを消去するか指定します．
<table><tr><th>設定値</th><th>概要</th></tr><tr><td>0</td><td>消去しない</td></tr></table></td><td>0-1</td></tr>
</table>

オプション	必須	説明		値範囲
		1	消去する	

使用例

```
//バッチNo.1からプロファイルNo.1のプロファイルデータを1つ取得する変数を追加
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("SPEC_BATCHPROFILE01","ProfileC
ount=1,BatchNo=1,ProfileNo=100");

//プロファイルデータを取得する
object[] retValue = var.Value as object[];

object[] profileInfo = retValue[0] as object[]; // プロファイル情報

byte? ProfileNum = profileInfo [0] as byte?; // プロファイル数
byte? Luminication = profileInfo [1] as byte?; // 輝度情報
short? ProfileDataCount profileInfo [2] as short?; // プロファイルデータ点数
int? XPos = profileInfo [3] as int?; // X座標
int? XInterval = profileInfo [4] as int?; // X方向の間隔
uint? CurrentProfileNo = profileInfo [5] as uint?; // 最新のプロファイル番号
uint? OldestProfileNo = profileInfo [6] as uint?; // 最古のプロファイル番号
uint? GetProfileCount = profileInfo [7] as uint?; // 最古のバッチ内のプロファイル数
uint? GetOldestProfNo = profileInfo [8] as uint?; //読み出した中で最古のプロファイルNo.
uint? GetBatchNo = profileInfo [9] as uint?; //読み出したバッチNo.
uint? GetProfileNo = profileInfo [10] as uint?; //読み出したバッチ内のプロファイル数
uint? GetBatchProfileNo = profileInfo [11] as uint?; //何番目のプロファイルか
byte? GetProfileCount = profileInfo [12] as uint?; //読み出した中で最古のプロファイルNo.
byte? isBatchMesure = profileInfo [13] as byte?; //バッチ測定が終了しているか

int?[] profileData = retValue[1] as int[]?; // プロファイルデータ
```

2.3.1.8. COMMITED_BATCHPROFILE<??>

バッチ確定後最新からプロファイルデータを取得します.オプションで取得するプロファイルデータの数を指定し指定した数のプロファイルデータを取得します.

取得されるデータは 2.3.1.6 と同じになっておりますのでそちらを参照ください.

オプション	必須	説明	値範囲
TargetBank=[=<取得面>]	--	アクティブ面/非アクティブ面のどちらから取得するかを指定します. 設定値 / 概要 0 / アクティブ面 1 / 非アクティブ面 デフォルト:0	0-1

オプション	必須	説明	値範囲
ProfileCount [=<取得するプロファイル数>]	○	読み出すプロファイル数を指定します.	1-255
Erase=[<プロファイルデータ削除有無>]	--	読み出したプロファイルとそれ以前のプロファイルを消去するか指定します. 設定値 / 概要 0 / 消去しない 1 / 消去する	0-1

使用例

```
//プロファイルデータを１つ取得する変数を追加
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("COMMITED_BATCHPROFILE01","Prof
ileCount=1");

//プロファイルデータを取得する
object[] retValue = var.Value as object[];

object[] profileInfo = retValue[0] as object[]; // プロファイル情報

byte? ProfileNum = profileInfo [0] as byte?; // プロファイル数
byte? Luminication = profileInfo [1] as byte?; // 輝度情報
short? ProfileDataCount profileInfo [2] as short?; // プロファイルデータ点数
int? XPos = profileInfo [3] as int?; // X座標
int? XInterval = profileInfo [4] as int?; // X方向の間隔
uint? CurrentProfileNo = profileInfo [5] as uint?; // 最新のプロファイル番号
uint? OldestProfileNo = profileInfo [6] as uint?; // 最古のプロファイル番号
uint? GetProfileCount = profileInfo [7] as uint?; // 最古のバッチ内のプロファイル数
uint? GetOldestProfNo = profileInfo [8] as uint?; //読み出した中で最古のプロファイルNo.
uint? GetBatchNo = profileInfo [9] as uint?; //読み出したバッチNo.
uint? GetProfileNo = profileInfo [10] as uint?; //読み出したバッチ内のプロファイル数
uint? GetBatchProfileNo = profileInfo [11] as uint?; //何番目のプロファイルか
byte? GetProfileCount = profileInfo [12] as uint?; //読み出した中で最古のプロファイルNo.
byte? isBatchMesure = profileInfo [13] as byte?; //バッチ測定が終了しているか

int?[] profileData = retValue[1] as int[]?; // プロファイルデータ
```

2.3.1.9. CURRENTONLY_BATCHPROFILE<??>

プロファイルデータの最新のみ取得します.

取得されるデータは 2.3.1.6 と同じになっておりますのでそちらを参照ください.

オプション	必須	説明	値範囲
TargetBank=[=<取得面>]	--	アクティブ面/非アクティブ面のどちらから	0-1

<table>
<tr><th>オプション</th><th>必須</th><th>説明</th><th>値範囲</th></tr>
<tr><td></td><td></td><td>取得するかを指定します.
<table><tr><th>設定値</th><th>概要</th></tr><tr><td>0</td><td>アクティブ面</td></tr><tr><td>1</td><td>非アクティブ面</td></tr></table>デフォルト:0</td><td></td></tr>
<tr><td>Erase=[<プロファイルデータ削除有無>]</td><td>--</td><td>読み出したプロファイルとそれ以前のプロファイルを消去するか指定します.
<table><tr><th>設定値</th><th>概要</th></tr><tr><td>0</td><td>消去しない</td></tr><tr><td>1</td><td>消去する</td></tr></table></td><td>0-1</td></tr>
</table>

使用例

```
//プロファイルデータを１つ取得する変数を追加
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("CURRENTONLY_BATCHPROFILE01","");

//プロファイルデータを取得する
object[] retValue = var.Value as object[];

object[] profileInfo = retValue[0] as object[]; // プロファイル情報

byte? ProfileNum = profileInfo [0] as byte?; // プロファイル数
byte? Luminication = profileInfo [1] as byte?; // 輝度情報
short? ProfileDataCount profileInfo [2] as short?; // プロファイルデータ点数
int? XPos = profileInfo [3] as int?; // X座標
int? XInterval = profileInfo [4] as int?; // X方向の間隔
uint? CurrentProfileNo = profileInfo [5] as uint?; // 最新のプロファイル番号
uint? OldestProfileNo = profileInfo [6] as uint?; // 最古のプロファイル番号
uint? GetProfileCount = profileInfo [7] as uint?; // 最古のバッチ内のプロファイル数
uint? GetOldestProfNo = profileInfo [8] as uint?; //読み出した中で最古のプロファイルNo.
uint? GetBatchNo = profileInfo [9] as uint?; //読み出したバッチNo.
uint? GetProfileNo = profileInfo [10] as uint?; //読み出したバッチ内のプロファイル数
uint? GetBatchProfileNo = profileInfo [11] as uint?; //何番目のプロファイルか
byte? GetProfileCount = profileInfo [12] as uint?; //読み出した中で最古のプロファイルNo.
byte? isBatchMesure = profileInfo [13] as byte?; //バッチ測定が終了しているか

int?[] profileData = retValue[1] as int[]?; // プロファイルデータ
```

2.3.1.10. CURRENT_BATCHPROFILE_SIMPLE<??>

最新からプロファイルデータを取得します.オプションで取得するプロファイルデータの数を指定し指定した数のプロファイルデータを取得します.

取得されるデータを以下に示します.

型説明			
VT_ARRAY | VT_VARIANT			
0	VT_ARRAY | VT_VARIANT		
	0.0	VT_UI1	1 単位のデータに格納されるプロファイル数を取得します.
	0.1	VT_UI1	輝度の ON/OFF を取得します.
	0.2	VT_UI2	1 プロファイル中のデータ点数を取得します.
	0.3	VT_I4	1 点目の X 座標を取得します.
	0.4	VT_I4	データ点の X 方向の間隔
	0.5	VT_I4	取得時点の最新プロファイル No. を取得します.
	0.6	VT_UI4	コントローラが保持する最古のプロファイル No. を取得します.
	0.7	VT_UI4	コントローラが保持する最古のバッチ内のプロファイル数を取得します.
	0.8	VT_UI4	読み出した中で最古のプロファイル No. を取得します.
	0.9	VT_UI4	今回読み出したバッチ No. を取得します.
	0.10	VT_UI4	今回読み出したバッチ内のプロファイル数を取得します.
	0.11	VT_UI4	読み出したバッチ内の何番目のプロファイルか取得します.
	0.12	VT_UI1	今回読み出したプロファイル数を取得します.
	0.13	VT_UI1	最新バッチ測定が完了しているか取得します.
1.n	VT_ARRAY|VT_I4		読み出したプロファイルデータのヘッダ情報が取得されます.
2.n	VT_ARRAY|VT_I4		読み出したプロファイルデータの高さ情報が取得されます.
3.n	VT_ARRAY|VT_I4		読み出したプロファイルデータの輝度情報が取得されます.

オプション	必須	説明	値範囲
TargetBank=[=<取得面>]	--	アクティブ面/非アクティブ面のどちらから取得するかを指定します. 設定値 / 概要 0 / アクティブ面 1 / 非アクティブ面 デフォルト:0	0-1
ProfileCount [=<取得するプロファイル数>]	○	読み出すプロファイル数を指定します.	1-255

オプション	必須	説明	値範囲
Erase=[<プロファイルデータ削除有無>]	--	読み出したプロファイルとそれ以前のプロファイルを消去するか指定します. 設定値: 0 = 消去しない / 1 = 消去する	0-1

使用例

```
// プロファイルNo１０からプロファイル数を１つ取得する変数を追加
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("CURRENT_PROFILE_SIMPLE01","Pro
fileCount=1");

//プロファイルデータを取得する
object[] retValue = var.Value as object[];

object[] profileInfo = retValue[0] as object[]; // プロファイル情報

byte? ProfileNum = profileInfo [0] as byte?; // プロファイル数
byte? Luminication = profileInfo [1] as byte?; // 輝度情報
short? ProfileDataCount profileInfo [2] as short?; // プロファイルデータ点数
int? XPos = profileInfo [3] as int?; // X座標
int? XInterval = profileInfo [4] as int?; // X方向の間隔
uint? CurrentProfileNo = profileInfo [5] as uint?; // 最新のプロファイル番号
uint? OldestProfileNo = profileInfo [6] as uint?; // 最古のプロファイル番号
uint? GetProfileCount = profileInfo [7] as uint?; // 最古のバッチ内のプロファイル数
uint? GetOldestProfNo = profileInfo [8] as uint?; //読み出した中で最古のプロファイルNo.
uint? GetBatchNo = profileInfo [9] as uint?; //読み出したバッチNo.
uint? GetProfileNo = profileInfo [10] as uint?; //読み出したバッチ内のプロファイル数
uint? GetBatchProfileNo = profileInfo [11] as uint?; //何番目のプロファイルか
byte? GetProfileCount = profileInfo [12] as uint?; //読み出した中で最古のプロファイルNo.
byte? isBatchMesure = profileInfo [13] as byte?; //バッチ測定が終了しているか

int?[] profileData = retValue[1] as int[]?; // プロファイルヘッダーデータ
int?[] profileData = retValue[2] as int[]?; // プロファイル高さデータ
int?[] profileData = retValue[3] as int[]?; // プロファイル輝度データ
```

2.3.1.11. SPEC_BATCHPROFILE_SIMPLE <??>

位置指定した場所からプロファイルデータを取得します. オプションで取得するバッチ No とプロファイル No. を指定しその場所から指定した数のプロファイルデータを取得します.

取得されるデータは 2.3.1.10 と同じになっておりますのでそちらを参照ください.

オプション	必須	説明	値範囲
TargetBank=[=<取得面>]	--	アクティブ面/非アクティブ面のどちらから取得するかを指定します.	0-1

<table>
<tr><th>オプション</th><th>必須</th><th>説明</th><th>値範囲</th></tr>
<tr><td></td><td></td><td><table><tr><th>設定値</th><th>概要</th></tr><tr><td>0</td><td>アクティブ面</td></tr><tr><td>1</td><td>非アクティブ面</td></tr></table>デフォルト:0</td><td></td></tr>
<tr><td>BatchNo=[=<バッチ No>]</td><td>○</td><td>プロファイルデータを取得するバッチ No を指定します.</td><td>0~4294967295</td></tr>
<tr><td>ProfileNo=[=<取得開始するプロファイル No>]</td><td>○</td><td>プロファイル位置指定方法で 2 を指定した場合，取得対象のプロファイル No.を指定します.</td><td>0~4294967295</td></tr>
<tr><td>ProfileCount [=<取得するプロファイル数>]</td><td>○</td><td>読み出すプロファイル数を指定します.</td><td>1-255</td></tr>
<tr><td>Erase=[<プロファイルデータ削除有無>]</td><td>--</td><td>読み出したプロファイルとそれ以前のプロファイルを消去するか指定します.<table><tr><th>設定値</th><th>概要</th></tr><tr><td>0</td><td>消去しない</td></tr><tr><td>1</td><td>消去する</td></tr></table></td><td>0-1</td></tr>
</table>

使用例

```
//バッチ No.1 からプロファイル No.1 のプロファイルデータを 1 つ取得する変数を追加
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("SPEC_BATCHPROFILE_SIMPLE01","P
rofileCount=1,BatchNo=1,ProfileNo=100");

//プロファイルデータを取得する
object[] retValue = var.Value as object[];

object[] profileInfo = retValue[0] as object[]; // プロファイル情報

byte? ProfileNum = profileInfo [0] as byte?; // プロファイル数
byte? Luminication = profileInfo [1] as byte?; // 輝度情報
short? ProfileDataCount profileInfo [2] as short?; // プロファイルデータ点数
int? XPos = profileInfo [3] as int?; // X座標
int? XInterval = profileInfo [4] as int?; // X方向の間隔
uint? CurrentProfileNo = profileInfo [5] as uint?; // 最新のプロファイル番号
uint? OldestProfileNo = profileInfo [6] as uint?; // 最古のプロファイル番号
uint? GetProfileCount = profileInfo [7] as uint?; // 最古のバッチ内のプロファイル数
uint? GetOldestProfNo = profileInfo [8] as uint?; //読み出した中で最古のプロファイルNo.
uint? GetBatchNo = profileInfo [9] as uint?; //読み出したバッチNo.
uint? GetProfileNo = profileInfo [10] as uint?; //読み出したバッチ内のプロファイル数
uint? GetBatchProfileNo = profileInfo [11] as uint?; //何番目のプロファイルか
byte? GetProfileCount = profileInfo [12] as uint?; //読み出した中で最古のプロファイルNo.
```

```
byte? isBatchMesure = profileInfo [13] as byte?; //バッチ測定が終了しているか

int?[] profileData = retValue[1] as int[]?; // プロファイルヘッダーデータ
int?[] profileData = retValue[2] as int[]?; // プロファイル高さデータ
int?[] profileData = retValue[3] as int[]?; // プロファイル輝度データ
```

2.3.1.12. COMMITED_BATCHPROFILE_SIMPLE <??>

バッチ確定後最新からプロファイルデータを取得します.オプションで取得するプロファイルデータの数を指定し指定した数のプロファイルデータを取得します.

取得されるデータは 2.3.1.10 と同じになっておりますのでそちらを参照ください.

オプション	必須	説明	値範囲
TargetBank=[=<取得面>]	--	アクティブ面/非アクティブ面のどちらから取得するかを指定します. 設定値 / 概要 0 / アクティブ面 1 / 非アクティブ面 デフォルト:0	0-1
ProfileCount [=<取得するプロファイル数>]	○	読み出すプロファイル数を指定します.	1-255
Erase=[<プロファイルデータ削除有無>]	--	読み出したプロファイルとそれ以前のプロファイルを消去するか指定します. 設定値 / 概要 0 / 消去しない 1 / 消去する	0-1

使用例

```
//プロファイルデータを１つ取得する変数を追加
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("COMMITED_BATCHPROFILE_SIMPLE01
",,"ProfileCount=1");

//プロファイルデータを取得する
object[] retValue = var.Value as object[];

object[] profileInfo = retValue[0] as object[]; // プロファイル情報

byte? ProfileNum = profileInfo [0] as byte?; // プロファイル数
byte? Luminication = profileInfo [1] as byte?; // 輝度情報
short? ProfileDataCount profileInfo [2] as short?; // プロファイルデータ点数
int? XPos = profileInfo [3] as int?; // X座標
int? XInterval = profileInfo [4] as int?; // X方向の間隔
```

```
uint? CurrentProfileNo = profileInfo [5] as uint?; // 最新のプロファイル番号
uint? OldestProfileNo = profileInfo [6] as uint?; // 最古のプロファイル番号
uint? GetProfileCount = profileInfo [7] as uint?; // 最古のバッチ内のプロファイル数
uint? GetOldestProfNo = profileInfo [8] as uint?; //読み出した中で最古のプロファイルNo.
uint? GetBatchNo = profileInfo [9] as uint?; //読み出したバッチNo.
uint? GetProfileNo = profileInfo [10] as uint?; //読み出したバッチ内のプロファイル数
uint? GetBatchProfileNo = profileInfo [11] as uint?; //何番目のプロファイルか
byte? GetProfileCount = profileInfo [12] as uint?; //読み出した中で最古のプロファイルNo.
byte? isBatchMesure = profileInfo [13] as byte?; //バッチ測定が終了しているか

int?[] profileData = retValue[1] as int[]?; // プロファイルヘッダーデータ
int?[] profileData = retValue[2] as int[]?; // プロファイル高さデータ
int?[] profileData = retValue[3] as int[]?; // プロファイル輝度データ
```

2.3.1.13. CURRENTONLY_BATCHPROFILE_SIMPLE <??>

プロファイルデータの最新のみ取得します.

取得されるデータは 2.3.1.10 と同じになっておりますのでそちらを参照ください.

<table>
<tr><th>オプション</th><th>必須</th><th>説明</th><th>値範囲</th></tr>
<tr><td>TargetBank=[=<取得面>]</td><td>--</td><td>アクティブ面/非アクティブ面のどちらから取得するかを指定します.
<table><tr><th>設定値</th><th>概要</th></tr><tr><td>0</td><td>アクティブ面</td></tr><tr><td>1</td><td>非アクティブ面</td></tr></table>デフォルト:0</td><td>0-1</td></tr>
<tr><td>Erase=[<プロファイルデータ削除有無>]</td><td>--</td><td>読み出したプロファイルとそれ以前のプロファイルを消去するか指定します.
<table><tr><th>設定値</th><th>概要</th></tr><tr><td>0</td><td>消去しない</td></tr><tr><td>1</td><td>消去する</td></tr></table></td><td>0-1</td></tr>
</table>

使用例

```
//プロファイルデータを１つ取得する変数を追加
ORiN2.ManagedCAO.CCaoVariable var =controller.AddVariable("CURRENTONLY_BATCHPROFILE_SIMPLE01","");

//プロファイルデータを取得する
object[] retValue = var.Value as object[];

object[] profileInfo = retValue[0] as object[]; // プロファイル情報

byte? ProfileNum = profileInfo [0] as byte?; // プロファイル数
byte? Luminication = profileInfo [1] as byte?; // 輝度情報
```

```
short? ProfileDataCount profileInfo [2] as short?; // プロファイルデータ点数
int? XPos = profileInfo [3] as int?; // X座標
int? XInterval = profileInfo [4] as int?; // X方向の間隔
uint? CurrentProfileNo = profileInfo [5] as uint?; // 最新のプロファイル番号
uint? OldestProfileNo = profileInfo [6] as uint?; // 最古のプロファイル番号
uint? GetProfileCount = profileInfo [7] as uint?; // 最古のバッチ内のプロファイル数
uint? GetOldestProfNo = profileInfo [8] as uint?; //読み出した中で最古のプロファイルNo.
uint? GetBatchNo = profileInfo [9] as uint?; //読み出したバッチNo.
uint? GetProfileNo = profileInfo [10] as uint?; //読み出したバッチ内のプロファイル数
uint? GetBatchProfileNo = profileInfo [11] as uint?; //何番目のプロファイルか
byte? GetProfileCount = profileInfo [12] as uint?; //読み出した中で最古のプロファイルNo.
byte? isBatchMesure = profileInfo [13] as byte?; //バッチ測定が終了しているか

int?[] profileData = retValue[1] as int[]?; // プロファイルヘッダーデータ
int?[] profileData = retValue[2] as int[]?; // プロファイル高さデータ
int?[] profileData = retValue[3] as int[]?; // プロファイル輝度データ
```

2.4. 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.

メッセージ番号の XX の XX=DeviceID となっています. AddController 時に指定した DeviceID オプションの値が 1 の場合メッセージ番号は 1 となります.

メッセージの内容については, Value プロパティで取得することが可能です.

メッセージ番号	説明	参照
XX(0~127)	プロファイル情報をメッセージとして発行します.	P.44
1XX(256~383)	StartHighSpeedDataCommunication または AddController 時の HSDComm オプションで指定したプロファイル数を受信した際にメッセージを発行します.	P.44
2XX(512~639)	StartHighSpeedDataCommunicationSimpleArray で指定したプロファイル数を受信した際にメッセージを発行します.	P.44
3XX(768~895)	高速データ通信の終了メッセージを発行します. このメッセージは,AddController 時に HSDCommEndMess オプションを有効にした場合, 発行されます.	P.45

2.4.1. プロファイル情報メッセージ

StartHighSpeedDataCommunication または AddController 時の HSDComm オプションで指定したプロファイル数を受信した際にメッセージを発行します. 以下にメッセージの内容を示します.

データ型

項目	型説明		
メッセージ概要	VT_ARRAY | VT_VARIANT		
	0.0	VT_UI1	1 単位のデータに格納されるプロファイル数を取得します.
	0.1	VT_UI1	輝度の ON/OFF を取得します.
	0.2	VT_UI2	1 プロファイル中のデータ点数を取得します.
	0.3	VT_I4	1 点目の X 座標を取得します.
	0.4	VT_I4	データ点の X 方向の間隔を取得します.

2.4.2. プロファイルデータメッセージ

プロファイルの情報を発行します. 本イベントは, プロファイルデータを受信した際に同時に発行されます. 以下にメッセージの内容を示します. メッセージ番号は, 0x100(256) + DeviceId になります. 2.2.1.1 で指定した DeviceId が 10 の場合は, 0x10A(266)がメッセージ番号となります.

データ型

項目	型説明		
メッセージ概要	VT_ARRAY | VT_VARIANT		
	n	VT_ARRAY|VT_I4	プロファイルデータを取得します. 取得したプロファイルデータが順に格納されます.

2.4.3. SimpleArray プロファイル情報メッセージ

StartHighSpeedDataCommunicationSimpleArray コマンド実行中に返却されるプロファイルの情報を発行します. 本イベントは, プロファイルデータを受信した際に同時に発行されます. 以下にメッセージの内容を示します.

データ型

項目	型説明		
メッセージ概要	0.n	VT_ARRAY|VT_I4	各プロファイルデータのヘッダ情報を取得します.
	1.n	VT_ARRAY|VT_UI2	読み出したプロファイルデータのうち高さ情報を取得します.
	2.n	VT_ARRAY|VT_UI2	読み出したプロファイルデータのうち輝度データを取得します.

2.4.4. 高速データ通信終了メッセージ

高速データ通信が終了した際にメッセージとして発行します. 本イベントは, AddController 時に HSDCommEndMess オプションを有効にした場合にのみ発行されます. 以下にメッセージの内容を示します.

データ型

<table>
<tr><th>項目</th><th>型説明</th></tr>
<tr><td>メッセージ概要</td><td>高速データ通信の終了事象の値を取得します.
以下に値とその内容を示します.
<table>
<tr><th>値</th><th>内容</th></tr>
<tr><td>1</td><td>StopHighSpeedDataCommunication コマンドにより高速データ通信が終了しました.</td></tr>
<tr><td>2</td><td>設定が変更されたため高速データ通信が終了されました.</td></tr>
<tr><td>4</td><td>実行プログラムが切り替えられたため高速データ通信が終了しました.</td></tr>
<tr><td>8</td><td>高速データ通信が強制的に停止されました.</td></tr>
<tr><td>256</td><td>メモリークリアにより高速データ通信が終了しました.</td></tr>
</table></td></tr>
</table>

3. LJ-X8000A プロバイダによるプログラミング

LJ-X8000A プロバイダでは，以下の手順でクライアント PC と LJ-X8000A を接続することができます.

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

LJ-X8000A に接続した後は，CaoController の Execute メソッドを使用することで LJ-X8000A の情報にアクセスすることができます.

3.1. プロファイ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 LJ-X8000A からプロファイルデータを取得する方法について示します. 表 3-1 にサンプルプログラムの要件を，図 3-1 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 3-1 サンプルプログラムの要件

要件	説明
接続先	TCP/IP で接続する
	接続先 IP アドレスは 192.168.1.2
	接続先ポート番号は 24691
処理内容	LJ-X8000A からプロファイルデータを取得する
	取得したプロファイルデータの内容を項目ごとに分割する

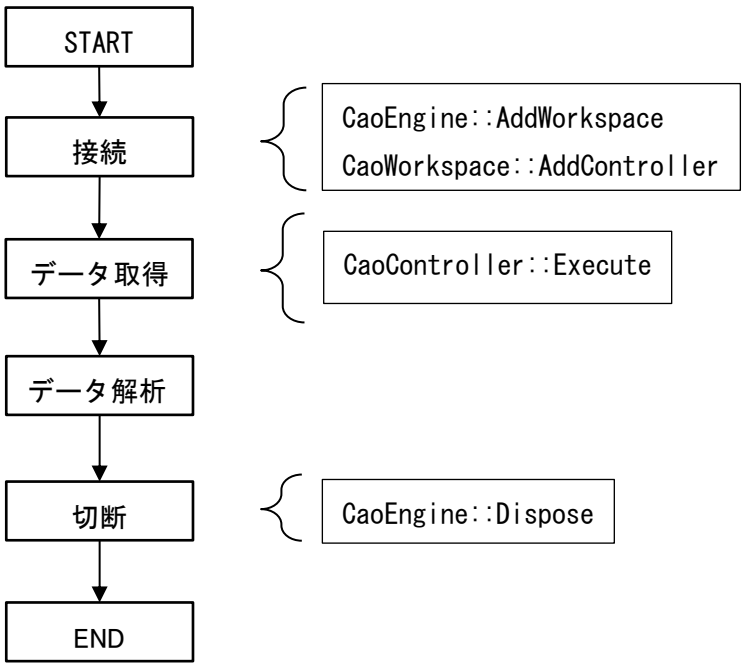


図 3-1 プロファイルデータ取得の流れ

以降の節から具体的なコードを示します.

3.1.1. 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 Sample.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;

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

        // プロファイルデータの取得
        object[] prm = new object[] { 0, 1, 1, 5, 1 };
        object[] retVarue = this.m_caoController.Execute("GetProfile", prm) as object[];

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

        byte? profCount = deviceparm[0] as byte?; //1単位のプロファイル数
        byte? luminanceOutput = deviceparm[1] as byte?;//輝度ON/OFF情報
        ushort? profDataCount = deviceparm[2] as ushort?; //1プロファイル中のデータ点数
        int? lXStart = deviceparm[3] as int?; // 1点目のX座標
```

```
        int? lXPitch = deviceparm[4] as int?; // データ点のX方向の間隔

        // 切断
        this.Disconnect();
}

// 接続メソッド
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("LJX8000A",
                           "CaoProv.KEYENCE.LJ-X8000",
                           "",
                           "Conn=ETH:127.0.0.1,DeviceID=1,Head=1, Timeout=1000");
}

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

3.1.1.1. 接続

LJ-X8000A と接続するためには，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．コントローラ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．CaoWorkpace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．以下にC#でのコード例を示します．

```
    // 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;
```

(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.

```
// 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.

```
// 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. LJ-X8000Aプロバイダでは, ETH接続とデバイス番号（任意）, ヘッダ番号をオプションで指定します. 以下にコード例を示します.

```
// CaoController オブジェクトの生成
this.m_caoController = this.m_caoWorkspace.AddController("LJX8000A",
                                "CaoProv.KEYENCE.LJ-X8000",
                                "",
                                "Conn=ETH:127.0.0.1,DeviceID=1,Head=1, Timeout=1000");
```

3.1.1.2. プロファイルデータの取得

Executeコマンドを使い, プロファイルデータの取得を行います. パラメータの各項目内容に関しては2.2.2.4.15を参照してください. 今回はプロファイル情報を先頭から5つ取得するように設定しています. 取得後は各データを別々の変数に格納します.

```
// パラメータ指定
object[] prm = new object[] { 0, 1, 1, 5, 1 };
// プロファイルデータの取得
object[] retVarue = this.m_caoController.Execute("GetProfile", prm) as object[];

object[] Profparm = retVarue[0] as object[];  //プロファイルの各設定項目
int?[] profileDataArray = retVarue[1] as int?[]; //5つの分プロファイルデータ

byte? profCount = Profparm [0] as byte?; //1単位のプロファイル数
byte? luminanceOutput = Profparm [1] as byte?;//輝度ON/OFF情報
ushort? profDataCount = Profparm [2] as ushort?; //1プロファイル中のデータ点数
int? lXStart = Profparm [3] as int?; // 1点目のX座標
int? lXPitch = Profparm [4] as int?; // データ点のX方向の間隔
```

3.1.1.3. 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．

```
// 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. LJ-X8000A プロバイダエラーコード

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 4-1 独自エラーコード表

エラー番号	説明
0x8010xxxx	ライブラリからのエラーとなります.

また，本プロバイダは，ライブラリのエラーコードを「0x8010****」でマスクして返します.
エラーコードの内容については「LJ-V8000A シリーズ 通信ライブラリ」を参照してください.

付録 A. API 対応表

LJX-8000A で使われているコマンドと LJX8IF.dll の API の対応表となります.

LJ-X8000A プロバイダコマンド	LJX8IF.dll 対応 API
GetError	LJX8IF_GetError
ClearError	LJX8IF_ClearError
TrgErrorReset	LJX8IF_TrgErrorReset
GetTriggerAndPulseCount	LJX8IF_GetTriggerAndPulseCount
GetHeadTemperture	LJX8IF_GetHeadTemperature
GetSerialNumber	LJX8IF_GetSerialNumber
GetAttentionStatus	LJX8IF_GetAttentionStatus
GetLedLightImage	LJX8IF_GetLedLightImage
Trigger	LJX8IF_Trigger
StartMesure	LJX8IF_StartMeasure
StopMesure	LJX8IF_StopMeasure
ClearMemory	LJX8IF_ClearMemory
GetActiveProgram	LJX8IF_GetActiveProgram
ChangeActiveProgram	LJX8IF_ChangeActiveProgram
GetProfile	LJX8IF_GetProfile
GetBatchProfile	LJX8IF_GetBatchProfile
GetbatchSimpleArray	LJX8IF_GetBatchSimpleArray
StartHighSpeedDataCommunication	LJX8IF_InitializeHighSpeedDataCommunication LJX8IF_PreStartHighSpeedDataCommunication LJX8IF_StartHighSpeedDataCommunication
StartHighSpeedDataCommunicationSimpleArray	LJX8IF_InitializeHighSpeedDataCommunicationSimpleArray LJX8IF_PreStartHighSpeedDataCommunication LJX8IF_StartHighSpeedDataCommunication
StopHighSpeedDataCommunication	LJX8IF_StopHighSpeedDataCommunication LJX8IF_FinalizeHighSpeedDataCommunication