# Open Protocol プロバイダ

## Atlas Copco 社製トルクコントローラ

Version 1.0.1

# ユーザーズ ガイド

October 30, 2018

【備考】

## 【改版履歴】

| バージョン | 日付 | 内容 |
|---|---|---|
| 1.0.0 | 2018-06-15 | 初版. |
| 1.0.1 | 2018-10-30 | メモリリーク バグ修正 |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

## 【対応機器】

| 機種 | バージョン | 注意事項 |
|---|---|---|
| PF4000 | 14.12.1 | Open Protocol 通信仕様にて動作確認 |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# 目次

# 1. はじめに

本書は，アトラスコプコ社製トルクコントローラの情報取得，イベント監視を行う CAO プロバイダのユーザーズガイドです．本書で扱う CAO プロバイダ(CaoProv.ATLASCOPCO.OPENPROTOCOL.dll)を Open Protocol プロバイダと呼びます．

第 2 章に Open Protocol プロバイダの概要，変数の詳細を記載しています．

Open Protocol プロバイダで実装している通信電文の詳細についてはアトラスコプコ社の“OpenProtocol_Specification_R_2_8_0_9836 4415 01”を参照してください．

# 2. プロバイダの概要

## 2.1. 概要

Open Protocol プロバイダは，アトラスコプコ社製トルクコントローラと Ethernet またはシリアル通信ポートを使用してデータの送受信を行う CAO プロバイダです．そのファイル形式は DLL(Dynamic Link Library)であり，CAO エンジンから使用時に動的にロードされます．Open Protocol プロバイダを使用するにあたっては ORiN2SDK をインストールするか，下表を参照して手作業でレジストリ登録を行う必要があります．

**表 2-1 Open Protocol プロバイダ**

| | |
|---|---|
| ファイル名 | CaoProvATLASCOPCOOPENPROTOCOL.dll |
| ProgID | CaoProv.ATLASCOPCO.OPENPROTOCOL |
| レジストリ登録 | regsvr32 CaoProvATLASCOPCOOPENPROTOCOL.dll |
| レジストリ登録の抹消 | regsvr32 /u CaoProvATLASCOPCOOPENPROTOCOL.dll |

## 2.2. メソッド・プロパティ

### 2.2.1. CaoWorkspace::AddController メソッド

Open Protocol プロバイダは Controller オブジェクトの生成時に，イーサネット(TCP)・シリアル通信の初期処理を行います．

また，レビジョン設定ファイルの絶対パスをオプション文字列に指定します．

書式　AddController(<bstrCtrlName:BSTR>,<bstrProvName:BSTR>,
<bstrPCName:BSTR>,<bstrOption:BSTR>))

bstrCtrlName : [in] コントローラ名
bstrProvName : [in] プロバイダ名．固定値 =" CaoProv.ATLASCOPCO.OPENPROTOCOL"
bstrPcName : [in] プロバイダの実行マシン名
bstrOption : [in] オプション文字列

以下にオプション文字列に指定するリストを示します．

**表 2-2 CaoWorkspace::AddController のオプション文字列**

| オプション | 説明 |
|---|---|
| Conn=<接続パラメータ> | 必須．通信形態とその接続パラメータを設定します． |
| ConnTimeOut[=<タイムアウト時間>] | TCP 通信におけるタイムアウト時間をミリ秒で設定します．<br>(デフォルト:5000)<br>イーサネット通信時のみ有効なオプションです． |
| Timeout[=<タイムアウト時間>] | 送受信におけるタイムアウト時間をミリ秒で設定します．<br>(デフォルト:3000) |
| Path[=<ファイルパス>] | レビジョン設定ファイルの絶対パスを設定します．<br>省略時は，レビジョン=1 でメッセージを送信します． |
| KeepAliveTime[=<送信間隔>] | 有効保持コマンドを送信する間隔をミリ秒で設定します．<br>(デフォルト:10000)<br>=0 を設定した場合は，有効保持コマンドを送信しません．<br>※トルクコントローラの通信タイムアウトは15秒です．トルクコントローラは15秒間メッセージが交換されなかったときは，接続が切断されたと判断して，終了します．トルクコントローラとの接続を継続するためには，15秒以内の間隔で送信しなければなりません． |

#### 2.2.1.1. 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(“[ ]”)内のパラメータは省略可能を示します. また, 各パラメータの解説中の下線部はオプション指定を省略した時のデフォルト値を示します.

**【Ethernet デバイス】**

“Conn=ETH:<Dest IP Address>:<Dest Port No>”

“Conn=TCP:<Dest IP Address>:<Dest Port No>”

< Dest IP Address > : 接続先の IP アドレス.

< Dest Port No > : 接続先のポート番号. 4545

**【シリアルデバイス】**

“Conn=COM:<COM Port>[:<BaudRate>]”

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

<BaudRate> : 通信速度.

2400, 4800, 9600, 19200, 38400, 57600, 115200

#### 2.2.1.2. Path オプション

各メッセージのレビジョンを xml ファイル（レビジョン設定ファイル）で一元管理します.

Path オプションには, レビジョン設定ファイルの絶対パスを指定します.

例） Path=C:¥Users¥xxx¥AppData¥Local¥Temp¥IoTDS¥MIDLists.xml

メッセージにはそれぞれレビジョンがあります. 同じメッセージでもレビジョンによってデータの内容が異なります. プロバイダは送信しようとするメッセージの最適なレビジョンを知りません. 送信者がメッセージ毎のレビジョンを把握しておく必要があります.

xml ファイルのフォーマットは以下のようになります.

```
<?xml version="1.0" encoding="UTF-8"?>
<MIDList>
    <MID id="0001" name="Application Communication start">
        <rev>4</rev>
    </MID>
    <MID id="0003" name="Application Communication stop">
        <rev>1</rev>
    </MID>
    <MID id="0010" name="Parameter set ID upload request">
        <rev>1</rev>
    </MID>
    <MID id="0018" name="Select Parameter set">
        <rev>1</rev>
    </MID>
    <MID id="0060" name="Last tightening result data subscribe">
        <rev>6</rev>
    </MID>
    <MID id="0062" name="Last tightening result data acknowledge">
        <rev>1</rev>
    </MID>
    <MID id="0063" name="Last tightening result data unsubscribe">
        <rev>1</rev>
    </MID>
</MIDList>
```

#### 2.2.1.3. サンプルプログラム

AddControllerのサンプルプログラムを以下に示します.

【Ethernet】

```
    HRESULT hr = S_OK;
    ICaoEngine* pEng = NULL;
    ICaoWorkspaces *pWss = NULL;
    ICaoWorkspace *pWs = NULL;
    ICaoController *pCtrl = NULL;

    // CaoEngine の生成
    hr = CoCreateInstance(CLSID_CaoEngine,
                          NULL,
                          CLSCTX_LOCAL_SERVER,
                          IID_ICaoEngine,
                          (void **)&pEng);
    if (FAILED(hr)) {
        goto EndProc;
    }
    // CaoWorkspace コレクションの取得
    hr = pEng->get_Workspaces(&pWss);
    if (FAILED(hr)) {
        goto EndProc;
    }
    // CaoWorkspace の取得
    hr = pWss->Item(CComVariant(0L), &pWs);
    if (FAILED(hr)) {
        goto EndProc;
    }

    // CaoController の生成
    hr = pWs->AddController(CComBSTR(L"ATLASCOPCO_TEST"),
                          CComBSTR(L"CaoProv.ATLASCOPCO.OPENPROTOCOL"),
                          CComBSTR(L""),
                          CComBSTR(L"Conn=ETH:192.168.1.100:4545, Path=C:¥Users¥xxx¥AppData
                          ¥Local¥Temp¥IoTDS¥MIDLists.xml"),
                          &pCtrl);
    if (FAILED(hr)) {
        goto EndProc;
    }

    // ここに必要な処理を入れる
    // 値の設定,取得など

EndProc:
    if (pCtrl) pCtrl->Release();
    if (pWs) pWs->Release();
    if (pWss) pWss->Release();
    if (pEng) pEng->Release();
```

【シリアル】

```
    HRESULT hr = S_OK;
    ICaoEngine* pEng = NULL;
    ICaoWorkspaces *pWss = NULL;
    ICaoWorkspace *pWs = NULL;
    ICaoController *pCtrl = NULL;

    // CaoEngine の生成
    hr = CoCreateInstance(CLSID_CaoEngine,
                          NULL,
                          CLSCTX_LOCAL_SERVER,
                          IID_ICaoEngine,
                          (void **)&pEng);
    if (FAILED(hr)) {
        goto EndProc;
    }
    // CaoWorkspace コレクションの取得
    hr = pEng->get_Workspaces(&pWss);
    if (FAILED(hr)) {
        goto EndProc;
    }
    // CaoWorkspace の取得
    hr = pWss->Item(CComVariant(0L), &pWs);
    if (FAILED(hr)) {
        goto EndProc;
    }

    // CaoController の生成
    hr = pWs->AddController(CComBSTR(L"ATLASCOPCO_TEST"),
                          CComBSTR(L"CaoProv.ATLASCOPCO.OPENPROTOCOL"),
                          CComBSTR(L""),
                          CComBSTR(L"Conn=COM:7, Path=C:¥Users¥xxx¥AppData¥Local¥Temp
                          ¥IoTDS¥MIDLists.xml"),
                          &pCtrl);
    if (FAILED(hr)) {
        goto EndProc;
    }

    // ここに必要な処理を入れる
    // 値の設定,取得など

EndProc:
    if (pCtrl) pCtrl->Release();
    if (pWs) pWs->Release();
    if (pWss) pWss->Release();
    if (pEng) pEng->Release();
```

### 2.2.2. CaoController::Execute メソッド

CaoController クラスの Execute メソッドは，メッセージの送受信を行います．第1引数にコマンド名，第2引数にコマンドのパラメータを指定します．

Open Protocol プロバイダで実装されているコマンドの詳細は「3．コマンドリファレンス」を参照してください．

書式　Execute(<bstrCommandName:VT_BSTR>[,<vntParam:VT_VARIANT>])

<bstrCommandName > : [in] コマンド名

<vntParam> : [in] パラメータ

#### 2.2.2.1. サンプルプログラム

Executeのサンプルプログラムを以下に示します．

例)パラメータセットを選択する．

```
HRESULT hr = S_OK;
ICaoEngine* pEng = NULL;
ICaoWorkspaces *pWss = NULL;
ICaoWorkspace *pWs = NULL;
ICaoController *pCtrl = NULL;
ICaoVariable *pVar = NULL;
CComVariant vntGet;

// CaoEngine の生成
hr = CoCreateInstance(CLSID_CaoEngine,
                      NULL,
                      CLSCTX_LOCAL_SERVER,
                      IID_ICaoEngine,
                      (void **)&pEng);
if (FAILED(hr)) {
    goto EndProc;
}

// CaoWorkspace コレクションの取得
hr = pEng->get_Workspaces(&pWss);
if (FAILED(hr)) {
    goto EndProc;
}

// CaoWorkspace の取得
hr = pWss->Item(CComVariant(0L), &pWs);
if (FAILED(hr)) {
    goto EndProc;
}
```

```
    // CaoController の生成
    hr = pWs->AddController(CComBSTR(L"ATLASCOPCO_TEST"),
                            CComBSTR(L"CaoProv.ATLASCOPCO.OPENPROTOCOL"),
                            CComBSTR(L""),
                            CComBSTR(L"Conn=ETH:192.168.1.100:4545, Path=C:¥Users¥xxx¥AppData
                            ¥Local¥Temp¥IoTDS¥MIDLists.xml"),
                            &pCtrl);
    if (FAILED(hr)) {
        goto EndProc;
    }

    // 通信開始
    vntRet.Clear();
    vntParam.Clear();
    hr = pCtrl->Execute(CComBSTR("CommStart"), vntParam, &vntRet);
    if (FAILED(hr)) {
        goto EndProc;
    }

    ////////////////////////////////////////////////////////////////////////////////////////////////
    // SelPSet
    // パラメータセットの選択
    ////////////////////////////////////////////////////////////////////////////////////////////////
    vntRet.Clear();
    vntParam.Clear();
    vntParam.vt = VT_UI4;
    vntParam.ulVal = 1;
    hr = pCtrl->Execute(CComBSTR("SelPSet"), vntParam, &vntRet);
    if (FAILED(hr)) {
        goto EndProc;
    }

EndProc:

    // 通信停止
    vntRet.Clear();
    vntParam.Clear();
    hr = pCtrl->Execute(CComBSTR("CommStop"), vntParam, &vntRet);
    if (FAILED(hr)) {
        goto EndProc;
    }

    if (pVar) pVar->Release();
    if (pCtrl) pCtrl->Release();
    if (pWs) pWs->Release();
    if (pWss) pWss->Release();
    if (pEng) pEng->Release();
```

### 2.2.3. CaoController::AddVariable メソッド

CaoController クラスの AddVariable メソッドは，変数にアクセスするためのメソッドです．

Open Protocol プロバイダで実装されている変数は表 2-3 を参照してください．

書式　AddVariable(<bstrVariableName:VT_BSTR>[,<bstrOption:VT_BSTR>])

<bstrVariableName> : [in] 変数名

<bstrOption> : [in] オプション文字列

#### 2.2.3.1. サンプルプログラム

AddVariableのサンプルプログラムを以下に示します．

例)受信スレッドの最終エラーコードを取得する．

```
HRESULT hr = S_OK;
ICaoEngine* pEng = NULL;
ICaoWorkspaces *pWss = NULL;
ICaoWorkspace *pWs = NULL;
ICaoController *pCtrl = NULL;
ICaoVariable *pVar = NULL;
CComVariant vntGet;

// CaoEngine の生成
hr = CoCreateInstance(CLSID_CaoEngine,
                      NULL,
                      CLSCTX_LOCAL_SERVER,
                      IID_ICaoEngine,
                      (void **)&pEng);
if (FAILED(hr)) {
    goto EndProc;
}

// CaoWorkspace コレクションの取得
hr = pEng->get_Workspaces(&pWss);
if (FAILED(hr)) {
    goto EndProc;
}

// CaoWorkspace の取得
hr = pWss->Item(CComVariant(0L), &pWs);
if (FAILED(hr)) {
    goto EndProc;
}
```

```
    // CaoController の生成
    hr = pWs->AddController(CComBSTR(L"ATLASCOPCO_TEST"),
                            CComBSTR(L"CaoProv.ATLASCOPCO.OPENPROTOCOL"),
                            CComBSTR(L""),
                            CComBSTR(L"Conn=ETH:192.168.1.100:4545, Path=C:¥Users¥xxx¥AppData
                            ¥Local¥Temp¥IoTDS¥MIDLists.xml"),
                            &pCtrl);
    if (FAILED(hr)) {
        goto EndProc;
    }

    // 変数の生成
    hr = pCtrl->AddVariable(CComBSTR(L"@LASTERROR_RCV"), CComBSTR(L""), &pVar);
    if (FAILED(hr)) {
        goto EndProc;
    }

    // 値の取得
    pVar->get_Value(&vntGet);

EndProc:
    if (pVar) pVar->Release();
    if (pCtrl) pCtrl->Release();
    if (pWs) pWs->Release();
    if (pWss) pWss->Release();
    if (pEng) pEng->Release();
```

### 2.2.4. CaoController::OnMessage イベント

トルクコントローラからのイベント通知を受信すると，CaoController クラスの OnMessage イベントが発生します．

Open Protocol プロバイダで実装されているイベントは表 4-1 を参照してください．

#### 2.2.4.1. サンプルプログラム

OnMessageイベントのサンプルプログラムを以下に示します.

```
private void CtrlLoad()
{
    if (!this.ctrlLoaded)
    {
        try
        {
            // CAO エンジン生成
            this.caoEng = new CaoEngine();
            this.caoWss = this.caoEng.Workspaces;
            this.caoWs = this.caoWss.Item(0);

            // コントローラコレクションの取得
            this.caoCtrls = this.caoWs.Controllers;

            // コントローラに接続
            this.caoCtrl = this.caoCtrls.Add(this.textBox_CtrlName.Text,
                                             this.textBox_ProvName.Text,
                                             this.textBox_MachineName.Text,
                                             this.textBox_Option.Text);

            // OnMessage イベントハンドラの登録
            this.caoCtrl.OnMessage += new _ICaoControllerEvents_OnMessageEventHandler
                                                                        (caoCtrl_OnMessage);

            this.CtrlEnable(true);

            // コントローラ用ウィンドウ表示
            this.Ctrl = new Form2();
            this.Ctrl.ShowDialog(this);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
}
```

```
void caoCtrl_OnMessage(CaoMessage pICaoMess)
{
    try
    {
        if ((pICaoMess.Number == 61) || (pICaoMess.Number == 900))
        {
            // 別スレッドのメソッド呼び出し
            SetMessageCallback SetMsg = new SetMessageCallback(SetText);
            Invoke(SetMsg, pICaoMess);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

// OnMessage 時の処理関数
private void SetText(CaoMessage pICaoMsg)
{
    // イベント受信時の処理を追加
}
```

## 2.3. 変数一覧

### 2.3.1. コントローラクラス

**表 2-3 コントローラクラス 変数一覧**

| 変数名 | データ型 | 説明 | 属性 | |
|---|---|---|---|---|
| | | | get | put |
| @MAKER_NAME | VT_BSTR | プロバイダ作成メーカー | ○ | － |
| @VERSION | VT_BSTR | プロバイダバージョン | ○ | － |
| @LASTERROR_RCV | VT_I4 | 受信スレッドの最終エラーコード | ○ | － |

## 2.4. エラーコード

Open Protocol プロバイダでは，以下の固有エラーコードが定義されています．また，ORiN2 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 2-4 固有エラーコード**

| エラー名 | エラー番号 | 説明 |
|---|---|---|
| コマンドエラー | 0x801001xx | ある理由のために要求が実行できなかったとき，トルクコントローラからのエラーコードを xx の箇所に入れて返します．エラーコードの内容については Open Protocol リファレンスマニュアルを参照してください． |
| 回線未接続エラー | 0x80100200 | 通信開始コマンド（CommStart）が実行されていない． |

# 3. コマンドリファレンス

## 3.1. コマンド一覧

**表 3-1 コマンド一覧**



## 3.2. コマンド詳細

# CommStart

**構文** *object*.CommStart

**引数** なし

**戻り値** VT_UI1 | VT_ARRAY : 通信開始応答

**説明** 通信開始要求を送信します.
正常終了の場合, 回線を接続します.
異常終了時のエラーコードは,「2.4 エラーコード」を参照してください.
通信開始応答の詳細については, Open Protocol のマニュアルを参照してください.

# CommStop

**構文** *object*.CommStop

**引数** なし

**戻り値** VT_I4 : 0

**説明** 通信停止要求を送信します.
正常終了の場合, 回線を切断します.
異常終了時のエラーコードは,「2.4 エラーコード」を参照してください.

# RegResults

**構文** *object*.RegResults

**引数** なし

**戻り値** VT_I4 : 0

**説明** 最終締付け結果データの登録を行います.
登録後に締付けを実行すると, トルクコントローラから最終締付け結果データアップロード応答(MID = 0061)のイベントを受信します.
異常終了時のエラーコードは,「2.4 エラーコード」を参照してください.
最終締付け結果データアップロード応答の詳細については, Open Protocol のマニュアルを参照してください.

# UnregResults

**構文** *object*.UnregResults

**引数** なし

**戻り値** VT_I4 : 0

**説明** 最終締付け結果データの登録を取り消します.
本コマンド実行後に締付けを実行しても, トルクコントローラから最終締付け結果データアップロード応答のイベントは発生しません.
異常終了時のエラーコードは,「2.4 エラーコード」を参照してください.

# SelPSet

**構文** *object*.SelPSet(<PSET>)

**引数** <PSET> = VT_UI4 ： パラメータセット番号

**戻り値** VT_I4 : 0

**説明** トルクコントローラのパラメータセット番号を引数で指定したパラメータセット番号に切り替えます.
指定するパラメータセット番号は, 現在トルクコントローラに存在するパラメータセットの番号でなければなりません.
異常終了時のエラーコードは,「2.4 エラーコード」を参照してください.

# RegTraceCurve_Lastest

**構文** *object*.RegTraceCurve_Lastest<TYPE>

**引数** <TYPE> = VT_BSTR ： 曲線タイプ

**戻り値** VT_I4 : 0

**説明** 指定した曲線タイプの直近の波形曲線データの登録を行います.
曲線タイプはビットで指定します.

| 曲線タイプ | ビット位置 |
|---|---|
| Angle trace | 1 |
| Torque trace | 2 |
| Current trace (※) | 3 |
| Gradient trace (※) | 4 |
| Stroke trace (※) | 5 |
| Force trace (※) | 6 |

※Current trace～Force trace は, 現在のバージョンでは未サポートです.
例）Angle と Torque の波形曲線データを取得する場合, TYPE=3 を指定します.
本コマンドの実行に成功すると, トルクコントローラから波形曲線データ(MID = 0900)のイベントを受信します.
異常終了時のエラーコードは,「2.4 エラーコード」を参照してください.

波形曲線データの詳細については, Open Protocol のマニュアルを参照してください.

# RegTraceCurve_FromId

**構文** *object*.RegTraceCurve_FromId<DATA>

**引数** <DATA> = VT_BSTR | VT_ARRAY : 曲線タイプ, 締付け ID

**戻り値** VT_I4 : 0

**説明** 指定した締付け ID 以降の曲線タイプの波形曲線データの登録を行います.
曲線タイプは,「RegTraceCurve_Lastest」の項を参照してください.
締付け ID は 1～4294967295 の範囲で指定します.
本コマンドの実行に成功すると, トルクコントローラから波形曲線データ(MID = 0900)のイベントを受信します. 締付け ID 以降のデータが複数個ある場合は, データ数分のイベントが発生します.
異常終了時のエラーコードは,「2.4 エラーコード」を参照してください.

# RegTraceCurve_FromTime

**構文** *object*.RegTraceCurve_FromTime<DATA>

**引数** <DATA> = VT_BSTR | VT_ARRAY : 曲線タイプ, タイムスタンプ

**戻り値** VT_I4 : 0

**説明** 指定したタイムスタンプ以降の曲線タイプの波形曲線データの登録を行います.
曲線タイプは,「RegTraceCurve_Lastest」の項を参照してください.
タイムスタンプは 19 バイトの文字列で指定します. (YYYY-MM-DD:HH:MM:SS)
本コマンドの実行に成功すると, トルクコントローラから波形曲線データ(MID = 0900)のイベントを受信します. タイムスタンプ以降のデータが複数個ある場合は, データ数分のイベントが発生します.
異常終了時のエラーコードは,「2.4 エラーコード」を参照してください.

# RegTraceCurve_SnapId

**構文** *object*.RegTraceCurve_SnapId<DATA>

**引数** <DATA> = VT_BSTR | VT_ARRAY : 曲線タイプ, 開始締付け ID, 終了締付け ID

**戻り値** VT_I4 : 0

**説明** 開始締付け Id から終了締付け Id までの曲線タイプの波形曲線データの登録を行います.
曲線タイプは,「RegTraceCurve_Lastest」の項を参照してください.
締付け ID は 1～4294967295 の範囲で指定します.
本コマンドの実行に成功すると, トルクコントローラから波形曲線データ(MID = 0900)のイベントを受信します.
異常終了時のエラーコードは,「2.4 エラーコード」を参照してください.

# RegTraceCurve_SnapTime

**構文** *object*.RegTraceCurve_SnapTime<DATA>

**引数** <DATA> = VT_BSTR | VT_ARRAY : 曲線タイプ, 開始時刻, 終了時刻

**戻り値** VT_I4 : 0

**説明** 開始時刻から終了時刻までの曲線タイプの波形曲線データの登録を行います.
曲線タイプは,「RegTraceCurve_Lastest」の項を参照してください.
開始時刻/終了時刻は 19 バイトの文字列で指定します. (YYYY-MM-DD:HH:MM:SS)
本コマンドの実行に成功すると, トルクコントローラから波形曲線データ(MID = 0900)のイベントを受信します.
異常終了時のエラーコードは,「2.4 エラーコード」を参照してください.

# UnregTraceCurve

**構文** *object*.UnregTraceCurve<TYPE>

**引数** <TYPE> = VT_BSTR ： 曲線タイプ

**戻り値** VT_I4 : 0

**説明** 指定した曲線タイプの波形曲線データの登録を取り消します.
曲線タイプは,「RegTraceCurve_Lastest」の項を参照してください.
異常終了時のエラーコードは,「2.4 エラーコード」を参照してください.

# 4. OnMessage イベント

## 4.1. イベント一覧

**表 4-1 イベント一覧**

| イベント | イベント番号<br>Number | Value | | 備考 |
|---|---|---|---|---|
| | | 型 | 値 | |
| 最終締付け結果データ | 61 | VT_UI1 \|<br>VT_ARRAY | 受信<br>データ | RegResults コマンド呼び出し,<br>締付け実行で発生 |
| 波形曲線データ | 900 | | | 以下のコマンド実行で発生<br>・RegTraceCurve_Lastest<br>・RegTraceCurve_FromId<br>・RegTraceCurve_FromTime<br>・RegTraceCurve_SnapId<br>・RegTraceCurve_SnapTime |

# 5. 付録

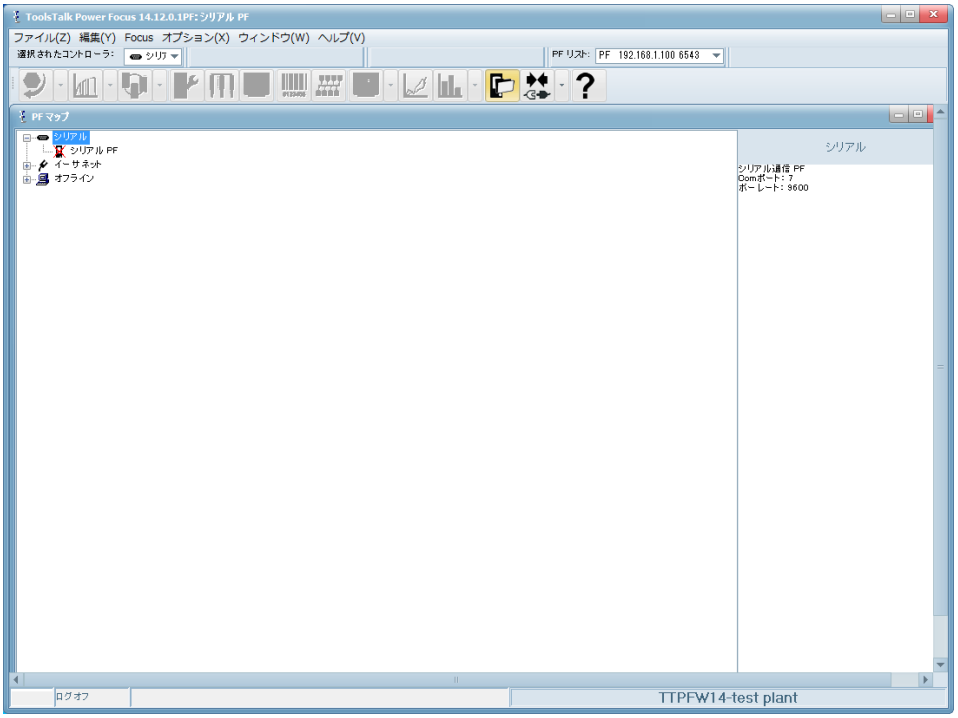
## 5.1. 通信の設定

Open Protocol プロバイダは, トルクコントローラと Ethernet またはシリアル通信ポートを使用してデータの送受信を行います.
各通信プロトコルの設定手順は以下のようになります.

【シリアル】(シリアル通信ポート 2 を使用)

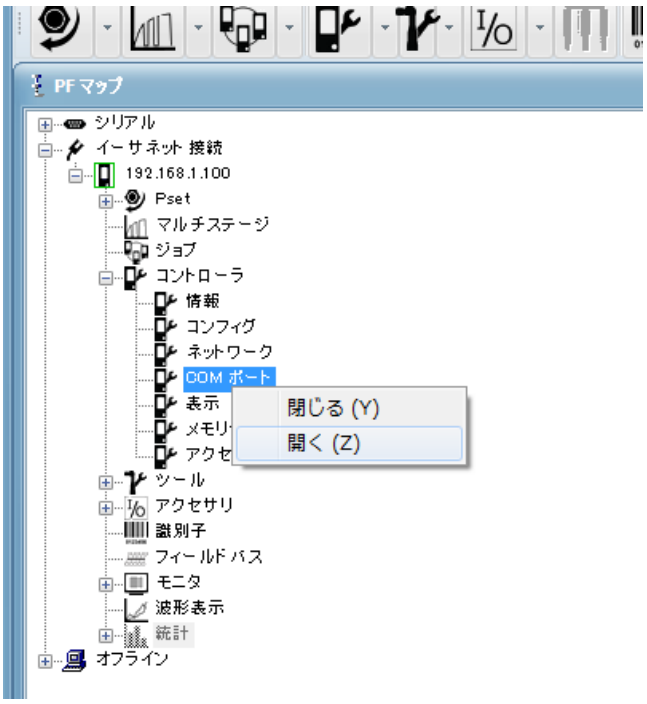
① PC とトルクコントローラを Ethernet で接続します.

② 設定ツール ToolsTalk Power Focus を起動します.

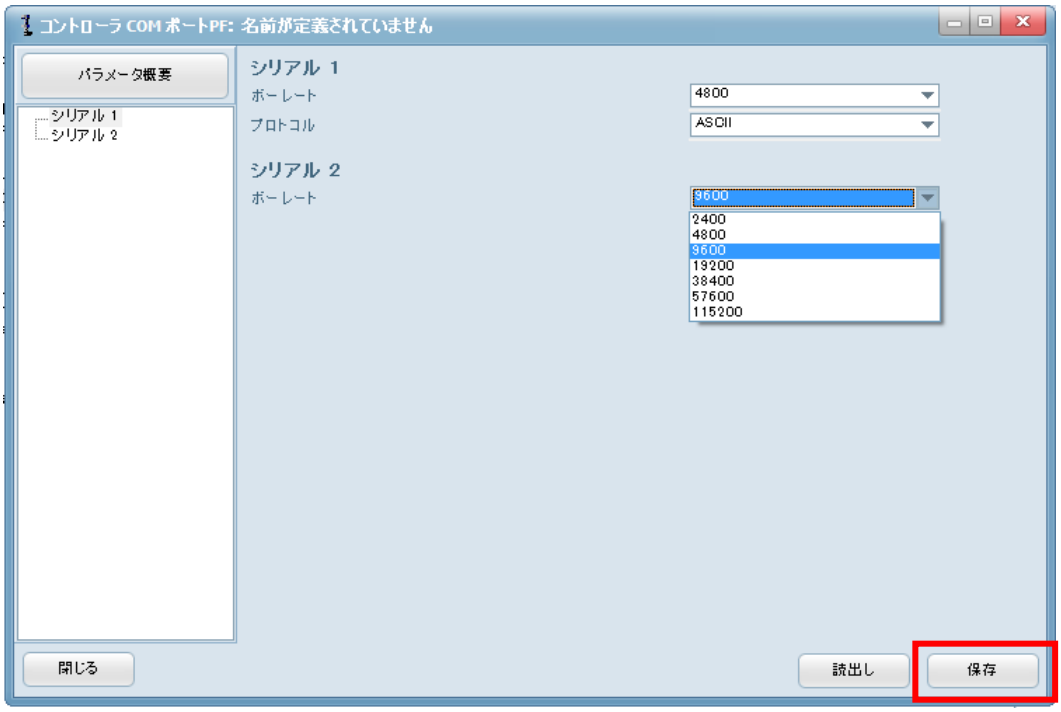



③ PF マップの「イーサネット PF」をダブルクリックして回線を接続します.

④ PF マップの「コントローラ」ツリーを開いて「COM ポート」を右クリックして「開く」を選択します.



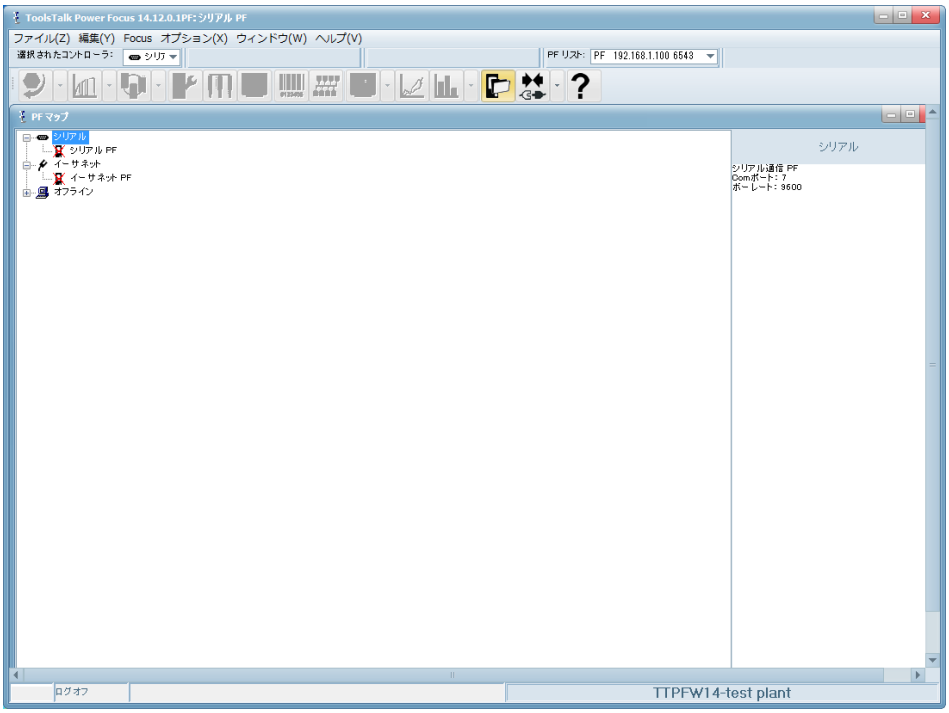

⑤ シリアル 2 のボーレートを設定して, 保存ボタンを押します.
トルクコントローラを再起動するようにメッセージが表示されますので, 再起動します. 再起動時にエラーが表示される場合は, しばらく時間をおいて電源再投入してください.



【Ethernet】

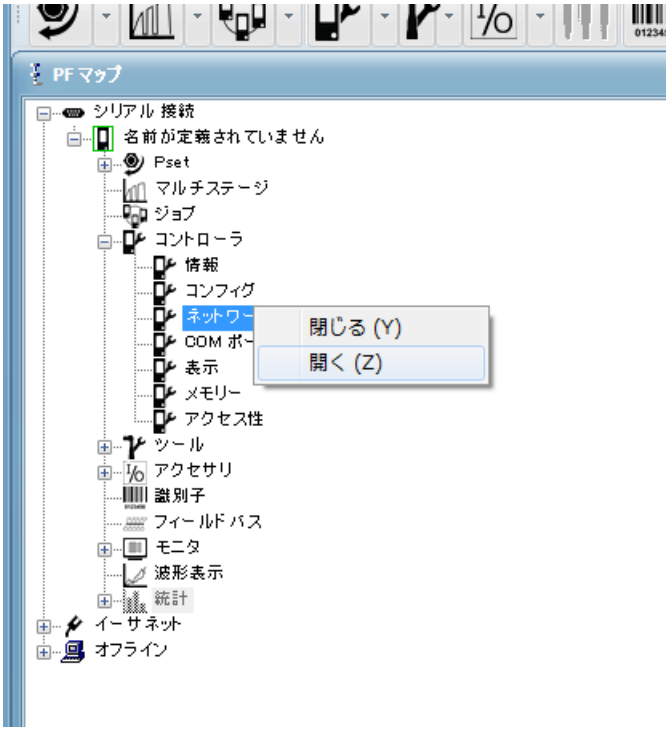
① PC とトルクコントローラをシリアルで接続します.

② 設定ツール ToolsTalk Power Focus を起動します.

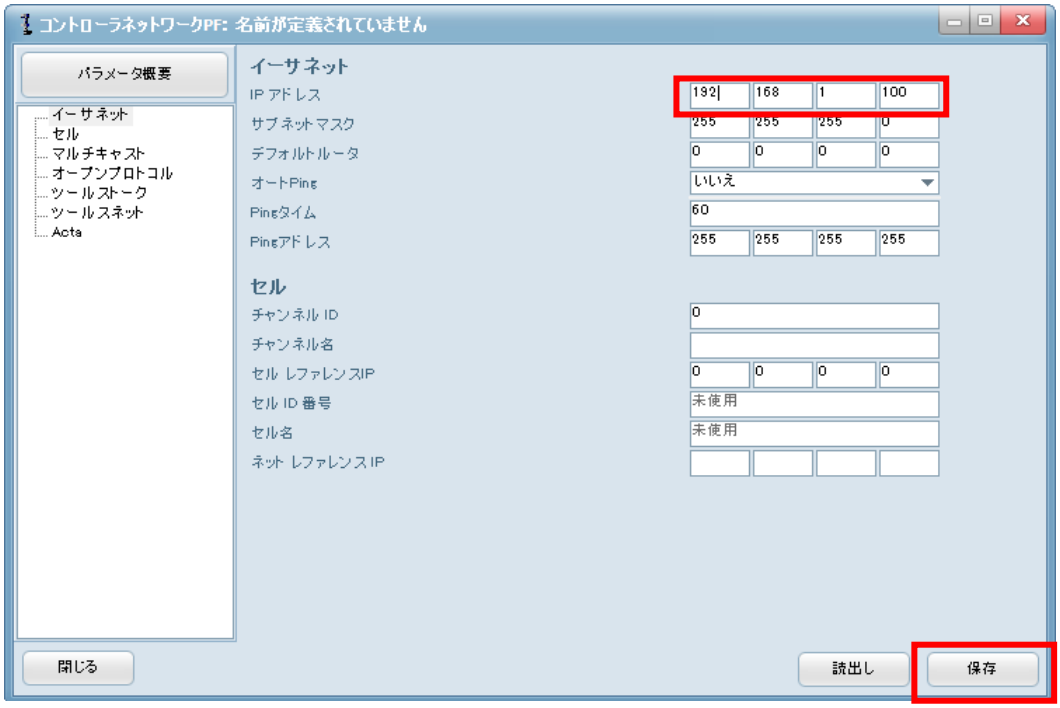



③ PF マップの「シリアル PF」をダブルクリックして回線を接続します.

④ PF マップの「コントローラ」ツリーを開いて「ネットワーク」を右クリックして「開く」を選択します.



⑤ IP アドレスを設定して，保存ボタンを押します．
トルクコントローラを再起動するようにメッセージが表示されますので，再起動します．再起動時にエラーが表示される場合は，しばらく時間をおいて電源再投入してください．




以上