アレイ 電子負荷装置 372XA シリーズ用

SCPI 通信

サンプルプログラム

取扱説明書

Ver.0.1.0

2013 年 12 月 26 日 作成

株式会社ティ・アンド・シー・テクニカル

はじめに

アレイ 電子負荷装置 372XA シリーズ SCPI 通信プログラムの
作成を支援するためのサンプルプログラムです。

SCPI コマンドを入力することで電子負荷装置へデータの書き込み・
読み出しが簡単に行えます。

ソースコードを公開しています。ご自由にお使いください。

このサンプルプログラムを使って何らかのソフトウェアを作成して
公開することや業務で使用することに特に制限はございません。

開発言語は Visual C++ 2012 です。

サンプルプログラムは通信を確立することに的を絞って
必要最小限のコードで作られています。

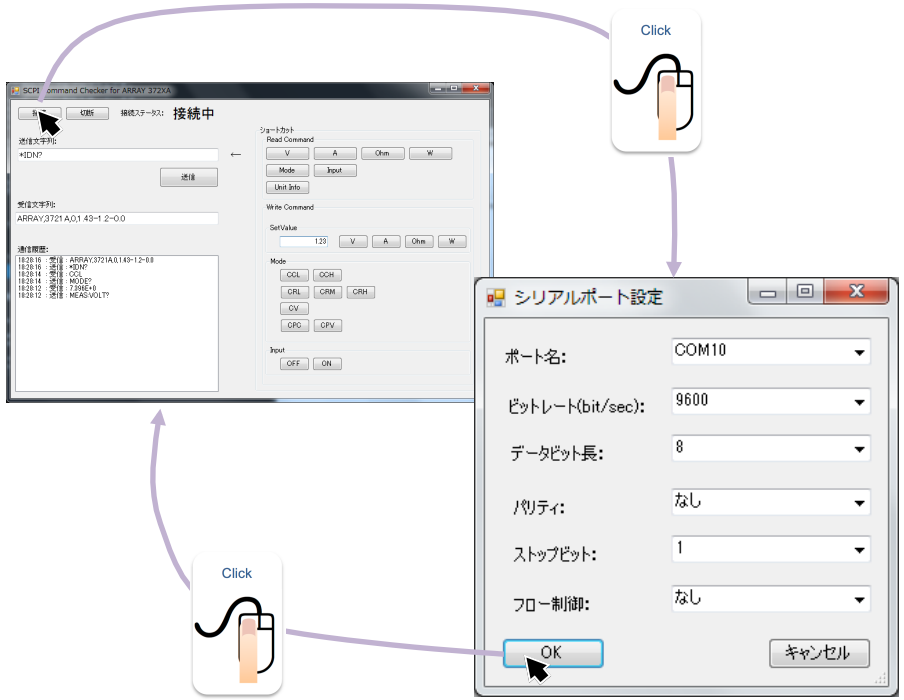
エラー判定やデータ変換、自動処理などは適宜、システムに合わせ作成して
組み入れてください。

接続する

シリアルポートに接続します。

ポート名はパソコンで使用する COM ポート番号を指定します。

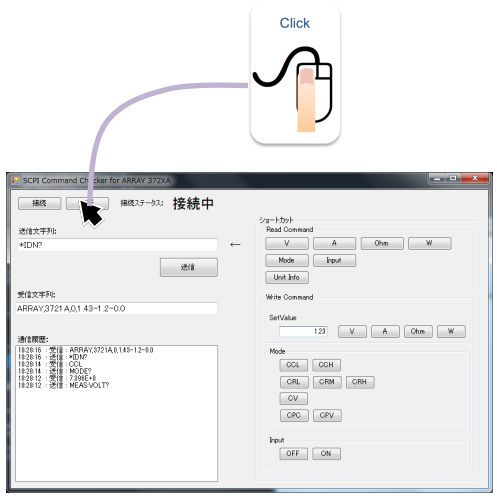
それ以外の通信設定は通信相手の設定と同じにします。



切断する

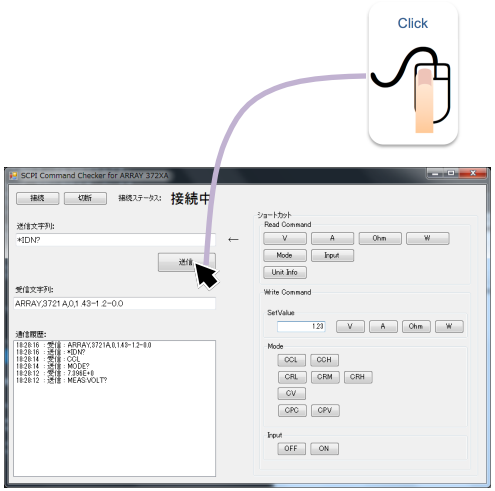
シリアルポートへの接続を切断します。

アプリケーションを終了する前に必ず切断してください。



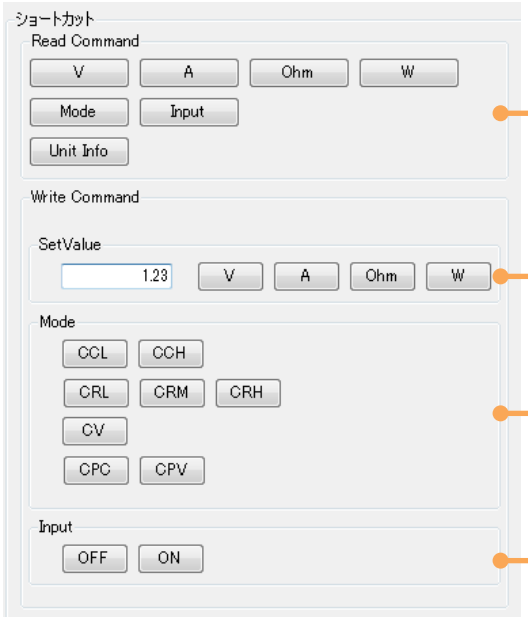
送信する

送信文字列欄に書かれたテキストデータを送信します。



画面右にある各ショートカットボタンを押すと

そのコマンドの文字列が送信文字列欄に簡単にセットできます。



選択されたデータを取得します。

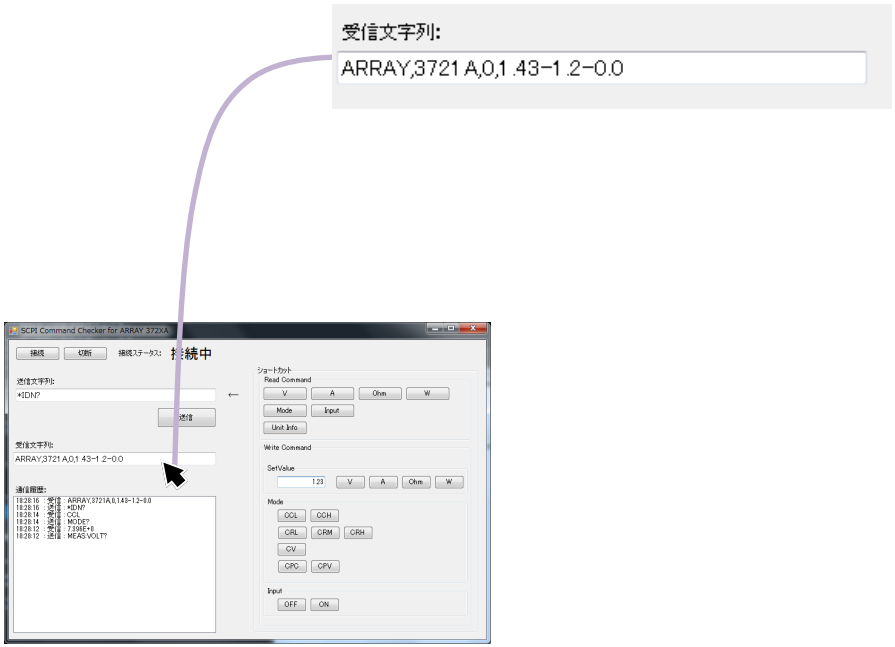
入力された値を書き込みます。

選択されたモードを書き込みます。

Input の ON/OFF を書き込みます。

受信する

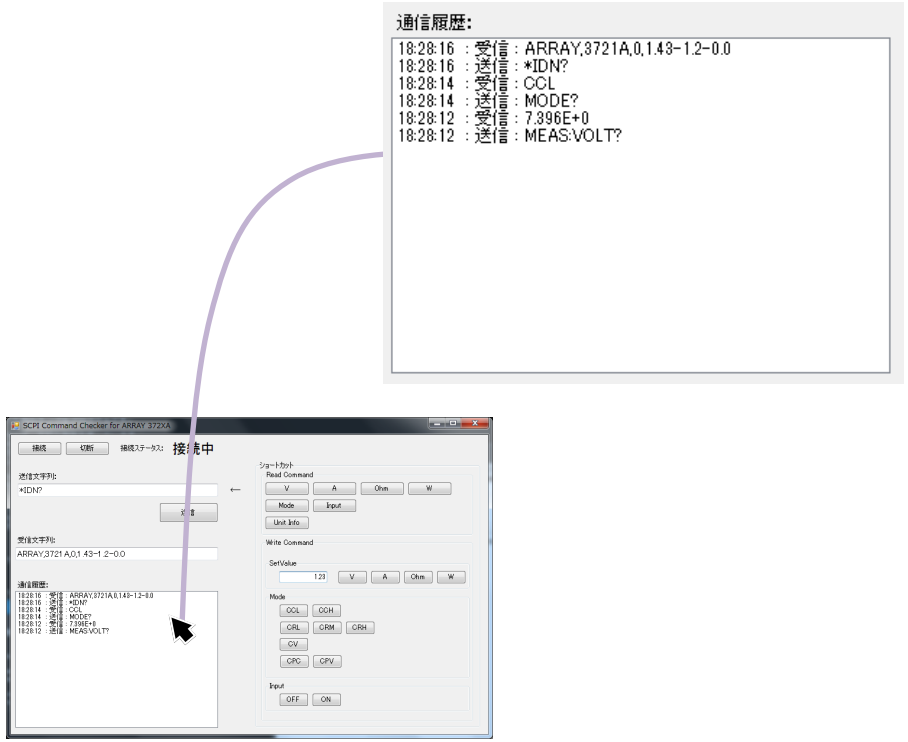
相手からデータを受信すると受信文字列欄にそのデータを表示します。



履歴を確認する

送信および受信したデータはリスト形式ですべて記録されます。

一番上に最新の記録が印字されます。



サンプルプログラムのコード

```
//接続ボタン
private: System::Void buttonConnect_Click(System::Object^   sender, System::EventArgs^   e) {
            formSerialConfiguration^ SerialConfiguration = gcnew formSerialConfiguration();
            if(serialPort->IsOpen == false) {       // ポートが閉じてることを確認
                SerialConfiguration->ShowDialog();              //フォームを表示する
                //フォームがキャンセルされたかどうかの確認
                if( SerialConfiguration->IsConfigurationEnabled == true){
                    try {
                        //設定されたデータの取得
                        serialPort->PortName = SerialConfiguration->GetSelectedPortName();
                        serialPort->BaudRate = SerialConfiguration->GetSelectedBaudRate();
                        serialPort->DataBits = SerialConfiguration->GetSelectedDataBits();
                        serialPort->Parity = SerialConfiguration->GetSelectedParity();
                        serialPort->StopBits = SerialConfiguration->GetSelectedStopBits();
                        serialPort->Handshake = SerialConfiguration->GetSelectedHandShake();

                        serialPort->Open();
                    } catch(Exception^ ex) {
                        System::Windows::Forms::MessageBox::Show(ex->Message);
                    }
                    if(serialPort->IsOpen == true) {            //ほんとに開かれたか確認
                        labelConnectionStatus->Text = "接続中";
                    }
                }
            }
}


//切断ボタン
private: System::Void buttonDisconnect_Click(System::Object^   sender, System::EventArgs^   e) {
            if(serialPort->IsOpen == true) {        //ほんとに開かれたか確認
                serialPort->Close();
            }

            if(serialPort->IsOpen == false) {       //ほんとに閉じたれたか確認
                labelConnectionStatus->Text = "切断中";
            }
}
```

```
//
delegate void SerialDataRecievedDelegate(String^ recvData);

private: void SerialDataRecieved(String^ recvData){
            //別スレッドで起動される関数。
            String^ s = recvData;
            textRecieved->Text =  s;
            String^ sTime = getDateToString();
            List1->Items->Insert(0,sTime + "  : 受信 : " + s);
}


//データ受信イベント
private: System::Void serialPort_DataReceived(System::Object^  sender,
    System::IO::Ports::SerialDataReceivedEventArgs^  e) {
            SerialDataRecievedDelegate^ dlgte = gcnew SerialDataRecievedDelegate( this,
                &SerialCommSample::Form1::SerialDataRecieved );
            String^ RecievedData = serialPort->ReadExisting();

            //1 バイトずつ受信することを前提
            if(serialPort->IsOpen == true) {        //シリアルポートが開いていることを確認してから。
                COMRevieveBuffer = COMRevieveBuffer + RecievedData; //受信バッファに追加
                //改行コードを検出する
                if(RecievedData == "¥r" ) {
                    RecievedData = RecievedData + "¥n";
                    COMRevieveBuffer = COMRevieveBuffer + RecievedData;
                    //受信した一行をテキストボックスに転送する。
                    //テキストボックスの操作は別スレッドの関数にする。
                    this->Invoke(dlgte, COMRevieveBuffer);
                    COMRevieveBuffer = "";
                }

                if(RecievedData == "¥n" ) {
                    RecievedData = RecievedData + "¥n";
                    COMRevieveBuffer = COMRevieveBuffer + RecievedData;
                    //受信した一行をテキストボックスに転送する。
                    //テキストボックスの操作は別スレッドの関数にする。
                    this->Invoke(dlgte, COMRevieveBuffer);
                    COMRevieveBuffer = "";
                }
            }
}


//送信ボタン
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
            if(serialPort->IsOpen == true) { //シリアルポートが開いていることを確認してから。
                String^ s = textTransmit->Text + "¥r¥n";
                serialPort->Write(s);
                String^ sTime = getDateToString();
                List1->Items->Insert(0,sTime + "  : 送信 : " + s);
            }
}
```

```
//読出ボタン 電圧
private: System::Void btnRead_Volt_Click(System::Object^  sender, System::EventArgs^  e) {
             textTransmit->Text = "MEAS:VOLT?";
}

//読出ボタン 電流
private: System::Void btnRead_Current_Click(System::Object^  sender, System::EventArgs^  e) {
              textTransmit->Text = "MEAS:CURR?";
}

//読出ボタン 抵抗
private: System::Void btnRead_Ohm_Click(System::Object^  sender, System::EventArgs^  e) {
             textTransmit->Text = "MEAS:RES?";
}

//読出ボタン 電力
private: System::Void btnRead_Watt_Click(System::Object^  sender, System::EventArgs^  e) {
             textTransmit->Text = "MEAS:POW?";
}

//読出ボタン モード
private: System::Void btnRead_Mode_Click(System::Object^  sender, System::EventArgs^  e) {
             textTransmit->Text =  "MODE?";
}

//読出ボタン Input On/Off
private: System::Void btnRead_Input_Click(System::Object^  sender, System::EventArgs^  e) {
             textTransmit->Text =  "INP?";
}

//読出ボタン ユニット情報
private: System::Void button2_Click(System::Object^  sender, System::EventArgs^  e) {
             textTransmit->Text = "*IDN?";
}

//書込ボタン 値セット 電圧
private: System::Void btnWrite_SetVolt_Click(System::Object^  sender, System::EventArgs^  e) {
             textTransmit->Text = "VOLT " + txtWrite_Value->Text;
}

//書込ボタン 値セット 電流
private: System::Void btnWrite_SetCurrent_Click(System::Object^  sender, System::EventArgs^  e) {
             textTransmit->Text = "CURR " + txtWrite_Value->Text;
}

//書込ボタン 値セット 抵抗
private: System::Void btnWrite_SetOhm_Click(System::Object^  sender, System::EventArgs^  e) {
             textTransmit->Text = "RES " + txtWrite_Value->Text;
}

//書込ボタン 値セット 電力
private: System::Void btnWrite_SetWatt_Click(System::Object^  sender, System::EventArgs^  e) {
             textTransmit->Text = "POW " + txtWrite_Value->Text;
}
```

```
//書込ボタン モード CCL
private: System::Void btnWrite_CCL_Click(System::Object^  sender, System::EventArgs^  e) {
            textTransmit->Text = "MODE CCL";
}

//書込ボタン モード CCH
private: System::Void btnWrite_CCH_Click(System::Object^  sender, System::EventArgs^  e) {
            textTransmit->Text = "MODE CCH";
}

//書込ボタン モード CRL
private: System::Void btnWrite_CRL_Click(System::Object^  sender, System::EventArgs^  e) {
            textTransmit->Text = "MODE CRL";
}

//書込ボタン モード CRM
private: System::Void btnWrite_CRM_Click(System::Object^  sender, System::EventArgs^  e) {
            textTransmit->Text = "MODE CRM";
}

//書込ボタン モード CRH
private: System::Void btnWrite_CRH_Click(System::Object^  sender, System::EventArgs^  e) {
            textTransmit->Text = "MODE CRH";
}

//書込ボタン モード CV
private: System::Void btnWrite_CV_Click(System::Object^  sender, System::EventArgs^  e) {
            textTransmit->Text = "MODE CV";
}

//書込ボタン モード CPC
private: System::Void btnWrite_CPC_Click(System::Object^  sender, System::EventArgs^  e) {
            textTransmit->Text = "MODE CPC";
}

//書込ボタン モード CPV
private: System::Void btnWrite_CPV_Click(System::Object^  sender, System::EventArgs^  e) {
            textTransmit->Text = "MODE CPV";
}

//書込ボタン インプット OFF
private: System::Void btnWrite_InputOFF_Click(System::Object^  sender, System::EventArgs^  e) {
            textTransmit->Text = "INP OFF";
}

//書込ボタン インプット ON

private: System::Void btnWrite_InputON_Click(System::Object^  sender, System::EventArgs^  e) {
            textTransmit->Text = "INP ON";
}
```

```
//現在時刻を文字列で取得
private: String^ getDateToString(){
		struct tm *date;
		time_t now;
		String^ s = "";
		time(&now);
		date = localtime(&now);

		s = date->tm_hour.ToString("00");
		s = s + ":" + date->tm_min.ToString("00");
		s = s + ":" + date->tm_sec.ToString("00");

		return s;
}


};
```