Oriental motor

αSTEP AZシリーズ miniドライバ

Ethernetタイプ

Modbus TCP 制御サンプルプログラム

目次

1　はじめに

■サンプルプログラムについて
プログラムはあまり複雑な構成にはせず、最低限必要な部分にしぼって、できるだけシンプルな形で作成しています。
送信データを変更する場合は、プログラムの中のレジスタアドレスやライト値を変更して試してください。

■プログラムや実行環境の技術的な内容について
プログラムや実行環境に関する技術的な内容については、参考書、または、インターネット上のフォーラムなどでご確認ください。

■動作確認環境
・確認ソフト：Visual Studio Code 1.83.0
・実行環境(Python)：Python 3.11.2

■製品の使い方について
製品の詳細については各シリーズのユーザーズマニュアルでご確認ください。

2　ドライバの設定

ドライバのIPアドレスとポート番号を設定します。
サンプルプログラムではドライバのIPアドレスを192.168.1.1、ポート番号を502としています。
ネットワーク内の通信機器のIPアドレスと重複しないように設定してください。
複数軸つなげるときもIPアドレスが重複しないように設定してください。

IPアドレスの設定方法はユーザーズマニュアルをご確認ください。

3　ソースコード（Python）

以下の内容がソースコードです。

```
from ctypes import BigEndianStructure
import socket
import array
import time
import sys
from socket import inet_aton

def main():
    # トランザクション ID 用カウンタ
    val = 0

    # IP アドレス
    address = "192.168.1.1"

    # ポート番号
    port = 502

    # バッファサイズ
    BUFSIZE = 4096

    # リモート I/O(R-OUT)から 28 レジスタの値を取得するクエリ
    frm_Mon =   [
                    0x00, 0x00,             # プロトコル ID :0x0000
                    0x00, 0x06,             # 伝文長 :6 byte
                    0x00,                   # ユニット ID :0
                    0x03,                   # FUNCTION CODE:0x03
                    0x01, 0x1C,             # レジスタアドレス:0x011C(リモート I/O R-OUT ~)
                    0x00, 0x1C              # 読み出し数 :28
                ]

    # 運転データの位置を書き換えるクエリ
    frm_WrOpeData_Pos = [
                            0x00, 0x00,         # プロトコル ID :0x0000
                            0x00, 0x2F,         # 伝文長 :47 byte
                            0x00,               # ユニット ID :0
                            0x10,               # FUNCTION CODE:0x10
                            0x01, 0x04,         # レジスタアドレス:0x0104 (リモート I/O R-IN ~)
                            0x00, 0x14,         # 書き込みレジスタ数 :20
                            0x28,               # バイト数 :40
                            0x00, 0x00,         # リモート I/O
                            0x00, 0x00,         # 運転データ No の選択
                            0x00, 0x00,         # 固定 I/O(IN)
                            0x00, 0x00,         # ダイレクトデータ運転　運転方式
                            0x00, 0x00,         # ダイレクトデータ運転　位置(下位)
                            0x00, 0x00,         # ダイレクトデータ運転　位置(上位)
                            0x00, 0x00,         # ダイレクトデータ運転　速度(下位)
                            0x00, 0x00,         # ダイレクトデータ運転　速度(上位)
                            0x00, 0x00,         # ダイレクトデータ運転　起動･変速レート(下位)
                            0x00, 0x00,         # ダイレクトデータ運転　起動･変速レート(上位)
                            0x00, 0x00,         # ダイレクトデータ運転　停止レート(下位)
                            0x00, 0x00,         # ダイレクトデータ運転　停止レート(上位)
                            0x00, 0x00,         # ダイレクトデータ運転　運転電流
                            0x00, 0x00,         # ダイレクトデータ運転　転送先
                            0x00, 0x00,         # 予約(reserved)
                            0x00, 0x00,         # リードパラメータ ID
                            0x00, 0x01,         # ライトリクエスト　 0x0001 (書き込み要求)
                            0x0C, 0x01,         # ライトパラメータ ID 0x0C01 (運転データ No.0 位置)
                            0x03, 0xE8,         # ライトデータ(下位) 1000 Step
                            0x00, 0x00          # ライトデータ(上位)
                        ]

    # ライトリクエストを OFF するクエリ
    frm_WrOpeData_Pos_WrRqoff = [
                            0x00, 0x00,         # プロトコル ID :0x0000
                            0x00, 0x2F,         # 伝文長 :47 byte
```

```
                        0x00,               # ユニット ID :0
                        0x10,               # FUNCTION CODE:0x10
                        0x01, 0x04,         # レジスタアドレス:0x0104 (リモート I/O R-IN ~)
                        0x00, 0x14,         # 書き込みレジスタ数 :20
                        0x28,               # バイト数 :40
                        0x00, 0x00,         # リモート I/O
                        0x00, 0x00,         # 運転データ No の選択
                        0x00, 0x00,         # 固定 I/O(IN)
                        0x00, 0x00,         # ダイレクトデータ運転 運転方式
                        0x00, 0x00,         # ダイレクトデータ運転 位置(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 位置(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 速度(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 速度(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 起動・変速レート(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 起動・変速レート(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 停止レート(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 停止レート(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 運転電流
                        0x00, 0x00,         # ダイレクトデータ運転 転送先
                        0x00, 0x00,         # 予約(reserved)
                        0x00, 0x00,         # リードパラメータ ID
                        0x00, 0x00,         # ライトリクエスト   0x0000
                        0x0C, 0x01,         # ライトパラメータ ID 0x0C01 (運転データ No.0 位置)
                        0x27, 0x10,         # ライトデータ(下位) 1000 Step
                        0x00, 0x00          # ライトデータ(上位)
                    ]

# 運転データの速度を書き換えるクエリ
frm_WrOpeData_Spd = [
                        0x00, 0x00,         # プロトコル ID :0x0000
                        0x00, 0x2F,         # 伝文長 :47 byte
                        0x00,               # ユニット ID :0
                        0x10,               # FUNCTION CODE:0x10
                        0x01, 0x04,         # レジスタアドレス:0x0104 (リモート I/O R-IN ~)
                        0x00, 0x14,         # 書き込みレジスタ数 :20
                        0x28,               # バイト数 :40
                        0x00, 0x00,         # リモート I/O
                        0x00, 0x00,         # 運転データ No の選択
                        0x00, 0x00,         # 固定 I/O(IN)
                        0x00, 0x00,         # ダイレクトデータ運転 運転方式
                        0x00, 0x00,         # ダイレクトデータ運転 位置(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 位置(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 速度(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 速度(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 起動・変速レート(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 起動・変速レート(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 停止レート(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 停止レート(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 運転電流
                        0x00, 0x00,         # ダイレクトデータ運転 転送先
                        0x00, 0x00,         # 予約(reserved)
                        0x00, 0x00,         # リードパラメータ ID
                        0x00, 0x01,         # ライトリクエスト   0x0001 (書き込み要求)
                        0x0C, 0x02,         # ライトパラメータ ID 0x0C02 (運転データ No.0 速度)
                        0x00, 0x64,         # ライトデータ(下位) 100 Hz
                        0x00, 0x00          # ライトデータ(上位)
                    ]

# ライトリクエストを OFF するクエリ
frm_WrOpeData_Spd_WrRqoff = [
                        0x00, 0x00,         # プロトコル ID :0x0000
                        0x00, 0x2F,         # 伝文長 :47 byte
                        0x00,               # ユニット ID :0
                        0x10,               # FUNCTION CODE:0x10
                        0x01, 0x04,         # レジスタアドレス:0x0104 (リモート I/O R-IN ~)
                        0x00, 0x14,         # 書き込みレジスタ数 :20
                        0x28,               # バイト数 :40
                        0x00, 0x00,         # リモート I/O
                        0x00, 0x00,         # 運転データ No の選択
                        0x00, 0x00,         # 固定 I/O(IN)
                        0x00, 0x00,         # ダイレクトデータ運転 運転方式
```

```
                        0x00, 0x00,         # ダイレクトデータ運転 位置(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 位置(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 速度(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 速度(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 起動・変速レート(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 起動・変速レート(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 停止レート(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 停止レート(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 運転電流
                        0x00, 0x00,         # ダイレクトデータ運転 転送先
                        0x00, 0x00,         # 予約(reserved)
                        0x00, 0x00,         # リードパラメータ ID
                        0x00, 0x00,         # ライトリクエスト   0x0000
                        0x0C, 0x02,         # ライトパラメータ ID 0x0C02 (運転データ No.0 速度)
                        0x00, 0x64,         # ライトデータ(下位) 100 Hz
                        0x00, 0x00          # ライトデータ(上位)
                    ]

# 固定 IO の START を ON するクエリ
frm_FixedIn_Start =     [
                        0x00, 0x00,         # プロトコル ID :0x0000
                        0x00, 0x2F,         # 伝文長 :47 byte
                        0x00,               # ユニット ID :0
                        0x10,               # FUNCTION CODE:0x10
                        0x01, 0x04,         # レジスタアドレス:0x0104 (リモート I/O R-IN ~)
                        0x00, 0x14,         # 書き込みレジスタ数 :20
                        0x28,               # バイト数 :40
                        0x00, 0x00,         # リモート I/O
                        0x00, 0x00,         # 運転データ No の選択
                        0x00, 0x08,         # 固定 I/O(IN)START ON
                        0x00, 0x00,         # ダイレクトデータ運転 運転方式
                        0x00, 0x00,         # ダイレクトデータ運転 位置(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 位置(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 速度(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 速度(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 起動・変速レート(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 起動・変速レート(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 停止レート(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 停止レート(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 運転電流
                        0x00, 0x00,         # ダイレクトデータ運転 転送先
                        0x00, 0x00,         # 予約(reserved)
                        0x00, 0x00,         # リードパラメータ ID
                        0x00, 0x00,         # ライトリクエスト
                        0x00, 0x00,         # ライトパラメータ ID
                        0x00, 0x00,         # ライトデータ(下位)
                        0x00, 0x00          # ライトデータ(上位)
                    ]


# 固定 IO の START を OFF するクエリ
frm_FixedIn_Stop =     [
                        0x00, 0x00,         # プロトコル ID :0x0000
                        0x00, 0x2F,         # 伝文長 :47 byte
                        0x00,               # ユニット ID :0
                        0x10,               # FUNCTION CODE:0x10
                        0x01, 0x04,         # レジスタアドレス:0x0104 (リモート I/O R-IN ~)
                        0x00, 0x14,         # 書き込みレジスタ数 :20
                        0x28,               # バイト数 :40
                        0x00, 0x00,         # リモート I/O
                        0x00, 0x00,         # 運転データ No の選択
                        0x00, 0x00,         # 固定 I/O(IN)START OFF
                        0x00, 0x00,         # ダイレクトデータ運転 運転方式
                        0x00, 0x00,         # ダイレクトデータ運転 位置(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 位置(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 速度(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 速度(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 起動・変速レート(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 起動・変速レート(上位)
                        0x00, 0x00,         # ダイレクトデータ運転 停止レート(下位)
                        0x00, 0x00,         # ダイレクトデータ運転 停止レート(上位)
```

```
                        0x00, 0x00,         # ダイレクトデータ運転 運転電流
                        0x00, 0x00,         # ダイレクトデータ運転 転送先
                        0x00, 0x00,         # 予約(reserved)
                        0x00, 0x00,         # リードパラメータ ID
                        0x00, 0x00,         # ライトリクエスト
                        0x00, 0x00,         # ライトパラメータ ID
                        0x00, 0x00,         # ライトデータ(下位)
                        0x00, 0x00          # ライトデータ(上位)
                    ]

    # ダイレクトデータ運転を実行するクエリ
    frm_ExeOpe =        [
                        0x00, 0x00,         # プロトコル ID :0x0000
                        0x00, 0x2F,         # 伝文長 :47 byte
                        0x00,               # ユニット ID :0
                        0x10,               # FUNCTION CODE:0x10
                        0x01, 0x04,         # レジスタアドレス:0x0104 (リモート I/O R-IN ~)
                        0x00, 0x14,         # 書き込みレジスタ数 :20
                        0x28,               # バイト数:40
                        0x00, 0x00,         # リモート I/O
                        0x00, 0x00,         # 運転データ No の選択
                        0x01, 0x00,         # 固定 I/O (IN) TRIG ON
                        0x00, 0x02,         # ダイレクトデータ運転 運転方式                :2 相対位置決め(指令
位置基準)
                        0x21, 0x34,         # ダイレクトデータ運転 位置 (下位)             :8500 step
                        0x00, 0x00,         # ダイレクトデータ運転 位置 (上位)
                        0x05, 0xDC,         # ダイレクトデータ運転 速度 (下位)             :2000 Hz
                        0x00, 0x00,         # ダイレクトデータ運転 速度 (上位)
                        0x07, 0xD0,         # ダイレクトデータ運転 起動・変速レート (下位)  :1.5 kHz/s
                        0x00, 0x00,         # ダイレクトデータ運転 起動・変速レート (上位)
                        0x05, 0xDC,         # ダイレクトデータ運転 停止レート (下位)       :1.5 kHz/s
                        0x00, 0x00,         # ダイレクトデータ運転 停止レート (上位)
                        0x03, 0xE8,         # ダイレクトデータ運転 運転電流                :100.0 %
                        0x00, 0x00,         # ダイレクトデータ運転 転送先
                        0x00, 0x00,         # 予約(reserved)
                        0x00, 0x00,         # リードパラメータ ID
                        0x00, 0x00,         # ライトリクエスト
                        0x00, 0x00,         # ライトパラメータ ID
                        0x00, 0x00,         # ライトデータ(下位)
                        0x00, 0x00          # ライトデータ(上位)
                    ]

    # ダイレクトデータ運転のトリガを OFF するクエリ
    frm_ExeOpe_TrgOFF =         [
                        0x00, 0x00,         # プロトコル ID :0x0000
                        0x00, 0x2F,         # 伝文長 :47 byte
                        0x00,               # ユニット ID :0
                        0x10,               # FUNCTION CODE:0x10
                        0x01, 0x04,         # レジスタアドレス:0x0104 (リモート I/O R-IN ~)
                        0x00, 0x14,         # 書き込みレジスタ数 :20
                        0x28,               # バイト数:40
                        0x00, 0x00,         # リモート I/O
                        0x00, 0x00,         # 運転データ No の選択
                        0x00, 0x00,         # 固定 I/O (IN) TRIG OFF
                        0x00, 0x02,         # ダイレクトデータ運転 運転方式                :2 相対位置決め(指令
位置基準)
                        0x21, 0x34,         # ダイレクトデータ運転 位置 (下位)             :8500 step
                        0x00, 0x00,         # ダイレクトデータ運転 位置 (上位)
                        0x05, 0xDC,         # ダイレクトデータ運転 速度 (下位)             :2000 Hz
                        0x00, 0x00,         # ダイレクトデータ運転 速度 (上位)
                        0x07, 0xD0,         # ダイレクトデータ運転 起動・変速レート (下位)  :1.5 kHz/s
                        0x00, 0x00,         # ダイレクトデータ運転 起動・変速レート (上位)
                        0x05, 0xDC,         # ダイレクトデータ運転 停止レート (下位)       :1.5 kHz/s
                        0x00, 0x00,         # ダイレクトデータ運転 停止レート (上位)
                        0x03, 0xE8,         # ダイレクトデータ運転 運転電流                :100.0 %
                        0x00, 0x00,         # ダイレクトデータ運転 転送先
                        0x00, 0x00,         # 予約(reserved)
                        0x00, 0x00,         # リードパラメータ ID
                        0x00, 0x00,         # ライトリクエスト
                        0x00, 0x00,         # ライトパラメータ ID
                        0x00, 0x00,         # ライトデータ(下位)
```

```
                        0x00, 0x00          # ライトデータ(上位)
                    ]

    print("**************************************")
    print("* Modbus TCP sample program for AZ series *")
    print("* Python *")
    print("* *")
    print("**************************************")
    print("\nDriver IP Address >" + address)
    print("\nDriver Port >" + str(port))

    # ソケット
    client  = socket.socket(socket.AF_INET,socket.SOCK_STREAM)  #TCP/IP

    # client  = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)  #UDP/IP

    # タイムアウトを設定
    client.settimeout(5)

    # ドライバと接続
    try:
        client.connect((address, int(port)))

    except OSError as msg:
        client.close()
        client = None

    # 接続失敗時は終了
    if client is None:
        print("could not open socket.")
        print("  IPAddress: " + address)
        print("  Port: " + str(port))
        input()
        sys.exit()

    frm_count = 0

    # 'q'または'Q'が入力されるまでループ
    while True:
        print("\n")
        print("1:Monitor")
        print("2:Set Operation Data")
        print("3:Operate")
        print("4:Execute Direct Data Operation")
        print("Q:Quit program")
        val = input()


        # Program Stop
        if val =='q' or val =='Q':
            break

        # モーターの検出位置を読み出し、ターミナルに表示する
        elif val == '1':
            # 送信用のクエリ作成
            frm_count_array = frm_count.to_bytes(2, "big")
            wkfrm = array.array('B', [])
            wkfrm.extend(frm_count_array)
            wkfrm.extend(frm_Mon)

            # クエリ送信
            client.sendto(wkfrm,(address,port))
            rcvData  = client.recv(BUFSIZE)
            print(rcvData)

            feedbackPos = int.from_bytes(array.array('B', [rcvData[19], rcvData[20], rcvData[17],
rcvData[18] ]), byteorder='big', signed=True)
            print(feedbackPos)
            frm_count += 1

        # 運転データ No.0 の位置、速度を書き換え
        elif val == '2':
            # 運転データ No.0 の位置
            # 送信用のクエリ作成
            frm_count_array = frm_count.to_bytes(2, "big")
            wkfrm = array.array('B', [])
```

```
        wkfrm.extend(frm_count_array)
        wkfrm.extend(frm_WrOpeData_Pos)

        # クエリ送信
        client.sendto(wkfrm,(address,port))
        rcvData  = client.recv(BUFSIZE)
        print(rcvData)
        frm_count += 1
        time.sleep(0.1)

        # ライトリクエストを OFF する
        # 送信用のクエリ作成
        frm_count_array = frm_count.to_bytes(2, "big")
        wkfrm = array.array('B', [])
        wkfrm.extend(frm_count_array)
        wkfrm.extend(frm_WrOpeData_Pos_WrRqoff)

        # クエリ送信
        client.sendto(wkfrm,(address,port))
        rcvData  = client.recv(BUFSIZE)
        print(rcvData)
        frm_count += 1
        time.sleep(0.1)

        # 運転データ No.0 の速度
        # 送信用のクエリ作成
        frm_count_array = frm_count.to_bytes(2, "big")
        wkfrm = array.array('B', [])
        wkfrm.extend(frm_count_array)
        wkfrm.extend(frm_WrOpeData_Spd)

        client.sendto(wkfrm,(address,port))
        rcvData  = client.recv(BUFSIZE)
        print(rcvData)
        frm_count += 1
        time.sleep(0.1)

        # ライトリクエストを OFF する
        # 送信用のクエリ作成
        frm_count_array = frm_count.to_bytes(2, "big")
        wkfrm = array.array('B', [])
        wkfrm.extend(frm_count_array)
        wkfrm.extend(frm_WrOpeData_Spd_WrRqoff)

        # クエリ送信
        client.sendto(wkfrm,(address,port))
        rcvData  = client.recv(BUFSIZE)
        print(rcvData)
        frm_count += 1


    # 固定 IO の START を ON する
    elif val == '3':
        # 固定 IO の START を ON するクエリ
        # 送信用のクエリ作成
        frm_count_array = frm_count.to_bytes(2, "big")
        wkfrm = array.array('B', [])
        wkfrm.extend(frm_count_array)
        wkfrm.extend(frm_FixedIn_Start)

        client.sendto(wkfrm,(address,port))
        rcvData  = client.recv(BUFSIZE)
        print(rcvData)
        frm_count += 1
        time.sleep(0.1)

        # 固定 IO の START を OFF するクエリ
        # 送信用のクエリ作成
        frm_count_array = frm_count.to_bytes(2, "big")
        wkfrm = array.array('B', [])
        wkfrm.extend(frm_count_array)
        wkfrm.extend(frm_FixedIn_Stop)

        client.sendto(wkfrm,(address,port))
        rcvData  = client.recv(BUFSIZE)
        print(rcvData)
```

```
            frm_count += 1

        # ダイレクトデータ運転を実行する
        elif val == '4':
            # ダイレクトデータ運転を実行するクエリ
            # 送信用のクエリ作成
            frm_count_array = frm_count.to_bytes(2, "big")
            wkfrm = array.array('B', [])
            wkfrm.extend(frm_count_array)
            wkfrm.extend(frm_ExeOpe)

            client.sendto(wkfrm,(address,port))
            rcvData  = client.recv(BUFSIZE)
            print(rcvData)
            frm_count += 1
            time.sleep(0.1)

            # ダイレクトデータ運転のトリガを OFF するクエリ
            # 送信用のクエリ作成
            frm_count_array = frm_count.to_bytes(2, "big")
            wkfrm = array.array('B', [])
            wkfrm.extend(frm_count_array)
            wkfrm.extend(frm_ExeOpe_TrgOFF)

            client.sendto(wkfrm,(address,port))
            rcvData  = client.recv(BUFSIZE)
            print(rcvData)
            frm_count += 1
        else:
            continue

        if frm_count > 65535:
            frm_count = 0

    client.close()

if __name__=="__main__":
    main()
```

4　ソースコード (C#)

以下の内容がソースコードです。

```
using System;
using System.Diagnostics.Metrics;
using System.Net;
using System.Net.Sockets;

namespace Modbus_Sample
{
    class Modbus_Sample
    {

        /// <summary>
        /// ドライバの IP アドレス :192.168.1.1
        /// </summary>
        public static readonly string Address = "192.168.1.1";

        /// <summary>
        /// ドライバの Port 番号 :502
        /// </summary>
        public static readonly int Port = 502;




        /// <summary>
        /// リモート I/O(R-OUT)から 28 レジスタの値を取得するクエリ
        /// </summary>
        public static byte[] frm_Mon = new byte[]
        {
            0x00, 0x00,             // プロトコル ID :0x0000
            0x00, 0x06,             // 伝文長 :6 byte
            0x00,                   // ユニット ID :0
            0x03,                   // FUNCTION CODE:0x03
            0x01, 0x1C,             // レジスタアドレス:0x011C(リモート I/O R-OUT ~)
            0x00, 0x1C              // 読み出し数 :28
        };

        /// <summary>
        /// 運転データの位置を書き換えるクエリ
        /// </summary>
        public static byte[] frm_WrOpeData_Pos = new byte[]
        {
            0x00, 0x00,         // プロトコル ID :0x0000
            0x00, 0x2F,         // 伝文長 :47 byte
            0x00,               // ユニット ID :0
            0x10,               // FUNCTION CODE:0x10
            0x01, 0x04,         // レジスタアドレス:0x0104 (リモート I/O R-IN ~)
            0x00, 0x14,         // 書き込みレジスタ数 :20
            0x28,               // バイト数 :40
            0x00, 0x00,         // リモート I/O
            0x00, 0x00,         // 運転データ No の選択
            0x00, 0x00,         // 固定 I/O(IN)
            0x00, 0x00,         // ダイレクトデータ運転 運転方式
            0x00, 0x00,         // ダイレクトデータ運転 位置(下位)
            0x00, 0x00,         // ダイレクトデータ運転 位置(上位)
            0x00, 0x00,         // ダイレクトデータ運転 速度(下位)
            0x00, 0x00,         // ダイレクトデータ運転 速度(上位)
            0x00, 0x00,         // ダイレクトデータ運転 起動・変速レート(下位)
            0x00, 0x00,         // ダイレクトデータ運転 起動・変速レート(上位)
            0x00, 0x00,         // ダイレクトデータ運転 停止レート(下位)
            0x00, 0x00,         // ダイレクトデータ運転 停止レート(上位)
            0x00, 0x00,         // ダイレクトデータ運転 運転電流
            0x00, 0x00,         // ダイレクトデータ運転 転送先
            0x00, 0x00,         // 予約(reserved)
            0x00, 0x00,         // リードパラメータ ID
            0x00, 0x01,         // ライトリクエスト   0x0001 (書き込み要求)
```

```
    0x0C, 0x01,         // ライトパラメータ ID 0x0C01 (運転データ No.0 位置)
    0x03, 0xE8,         // ライトデータ(下位) 1000 Step
    0x00, 0x00          // ライトデータ(上位)
};

/// <summary>
/// ライトリクエストを OFF するクエリ
/// </summary>
public static byte[] frm_WrOpeData_Pos_WrRqoff = new byte[]
{
    0x00, 0x00,         // プロトコル ID :0x0000
    0x00, 0x2F,         // 伝文長 :47 byte
    0x00,               // ユニット ID :0
    0x10,               // FUNCTION CODE:0x10
    0x01, 0x04,         // レジスタアドレス:0x0104 (リモート I/O R-IN ~)
    0x00, 0x14,         // 書き込みレジスタ数 :20
    0x28,               // バイト数 :40
    0x00, 0x00,         // リモート I/O
    0x00, 0x00,         // 運転データ No の選択
    0x00, 0x00,         // 固定 I/O(IN)
    0x00, 0x00,         // ダイレクトデータ運転 運転方式
    0x00, 0x00,         // ダイレクトデータ運転 位置(下位)
    0x00, 0x00,         // ダイレクトデータ運転 位置(上位)
    0x00, 0x00,         // ダイレクトデータ運転 速度(下位)
    0x00, 0x00,         // ダイレクトデータ運転 速度(上位)
    0x00, 0x00,         // ダイレクトデータ運転 起動・変速レート(下位)
    0x00, 0x00,         // ダイレクトデータ運転 起動・変速レート(上位)
    0x00, 0x00,         // ダイレクトデータ運転 停止レート(下位)
    0x00, 0x00,         // ダイレクトデータ運転 停止レート(上位)
    0x00, 0x00,         // ダイレクトデータ運転 運転電流
    0x00, 0x00,         // ダイレクトデータ運転 転送先
    0x00, 0x00,         // 予約(reserved)
    0x00, 0x00,         // リードパラメータ ID
    0x00, 0x00,         // ライトリクエスト   0x0000
    0x0C, 0x01,         // ライトパラメータ ID 0x0C01 (運転データ No.0 位置)
    0x27, 0x10,         // ライトデータ(下位) 1000 Step
    0x00, 0x00          // ライトデータ(上位)
};

/// <summary>
/// 運転データの速度を書き換えるクエリ
/// </summary>
public static byte[] frm_WrOpeData_Spd = new byte[]
{
    0x00, 0x00,         // プロトコル ID :0x0000
    0x00, 0x2F,         // 伝文長 :47 byte
    0x00,               // ユニット ID :0
    0x10,               // FUNCTION CODE:0x10
    0x01, 0x04,         // レジスタアドレス:0x0104 (リモート I/O R-IN ~)
    0x00, 0x14,         // 書き込みレジスタ数 :20
    0x28,               // バイト数 :40
    0x00, 0x00,         // リモート I/O
    0x00, 0x00,         // 運転データ No の選択
    0x00, 0x00,         // 固定 I/O(IN)
    0x00, 0x00,         // ダイレクトデータ運転 運転方式
    0x00, 0x00,         // ダイレクトデータ運転 位置(下位)
    0x00, 0x00,         // ダイレクトデータ運転 位置(上位)
    0x00, 0x00,         // ダイレクトデータ運転 速度(下位)
    0x00, 0x00,         // ダイレクトデータ運転 速度(上位)
    0x00, 0x00,         // ダイレクトデータ運転 起動・変速レート(下位)
    0x00, 0x00,         // ダイレクトデータ運転 起動・変速レート(上位)
    0x00, 0x00,         // ダイレクトデータ運転 停止レート(下位)
    0x00, 0x00,         // ダイレクトデータ運転 停止レート(上位)
    0x00, 0x00,         // ダイレクトデータ運転 運転電流
    0x00, 0x00,         // ダイレクトデータ運転 転送先
    0x00, 0x00,         // 予約(reserved)
    0x00, 0x00,         // リードパラメータ ID
    0x00, 0x01,         // ライトリクエスト   0x0001 (書き込み要求)
    0x0C, 0x02,         // ライトパラメータ ID 0x0C02 (運転データ No.0 速度)
    0x00, 0x64,         // ライトデータ(下位) 100 Hz
    0x00, 0x00          // ライトデータ(上位)
};
```

```
/// <summary>
/// ライトリクエストを OFF するクエリ
/// </summary>
public static byte[] frm_WrOpeData_Spd_WrRqoff = new byte[]
{
    0x00, 0x00,         // プロトコル ID :0x0000
    0x00, 0x2F,         // 伝文長 :47 byte
    0x00,               // ユニット ID :0
    0x10,               // FUNCTION CODE:0x10
    0x01, 0x04,         // レジスタアドレス:0x0104 (リモート I/O R-IN ~)
    0x00, 0x14,         // 書き込みレジスタ数 :20
    0x28,               // バイト数 :40
    0x00, 0x00,         // リモート I/O
    0x00, 0x00,         // 運転データ No の選択
    0x00, 0x00,         // 固定 I/O(IN)
    0x00, 0x00,         // ダイレクトデータ運転 運転方式
    0x00, 0x00,         // ダイレクトデータ運転 位置(下位)
    0x00, 0x00,         // ダイレクトデータ運転 位置(上位)
    0x00, 0x00,         // ダイレクトデータ運転 速度(下位)
    0x00, 0x00,         // ダイレクトデータ運転 速度(上位)
    0x00, 0x00,         // ダイレクトデータ運転 起動・変速レート(下位)
    0x00, 0x00,         // ダイレクトデータ運転 起動・変速レート(上位)
    0x00, 0x00,         // ダイレクトデータ運転 停止レート(下位)
    0x00, 0x00,         // ダイレクトデータ運転 停止レート(上位)
    0x00, 0x00,         // ダイレクトデータ運転 運転電流
    0x00, 0x00,         // ダイレクトデータ運転 転送先
    0x00, 0x00,         // 予約(reserved)
    0x00, 0x00,         // リードパラメータ ID
    0x00, 0x00,         // ライトリクエスト   0x0000
    0x0C, 0x02,         // ライトパラメータ ID 0x0C02 (運転データ No.0 速度)
    0x00, 0x64,         // ライトデータ(下位) 100 Hz
    0x00, 0x00          // ライトデータ(上位)
};

/// <summary>
/// 固定 IO の START を ON するクエリ
/// </summary>
public static byte[] frm_FixedIn_Start = new byte[]
{
    0x00, 0x00,         // プロトコル ID :0x0000
    0x00, 0x2F,         // 伝文長 :47 byte
    0x00,               // ユニット ID :0
    0x10,               // FUNCTION CODE:0x10
    0x01, 0x04,         // レジスタアドレス:0x0104 (リモート I/O R-IN ~)
    0x00, 0x14,         // 書き込みレジスタ数 :20
    0x28,               // バイト数 :40
    0x00, 0x00,         // リモート I/O
    0x00, 0x00,         // 運転データ No の選択
    0x00, 0x08,         // 固定 I/O(IN)START ON
    0x00, 0x00,         // ダイレクトデータ運転 運転方式
    0x00, 0x00,         // ダイレクトデータ運転 位置(下位)
    0x00, 0x00,         // ダイレクトデータ運転 位置(上位)
    0x00, 0x00,         // ダイレクトデータ運転 速度(下位)
    0x00, 0x00,         // ダイレクトデータ運転 速度(上位)
    0x00, 0x00,         // ダイレクトデータ運転 起動・変速レート(下位)
    0x00, 0x00,         // ダイレクトデータ運転 起動・変速レート(上位)
    0x00, 0x00,         // ダイレクトデータ運転 停止レート(下位)
    0x00, 0x00,         // ダイレクトデータ運転 停止レート(上位)
    0x00, 0x00,         // ダイレクトデータ運転 運転電流
    0x00, 0x00,         // ダイレクトデータ運転 転送先
    0x00, 0x00,         // 予約(reserved)
    0x00, 0x00,         // リードパラメータ ID
    0x00, 0x00,         // ライトリクエスト
    0x00, 0x00,         // ライトパラメータ ID
    0x00, 0x00,         // ライトデータ(下位)
    0x00, 0x00          // ライトデータ(上位)
};

/// <summary>
/// 固定 IO の START を OFF するクエリ
/// </summary>
```

```
public static byte[] frm_FixedIn_Stop = new byte[]
{
    0x00, 0x00,         // プロトコル ID :0x0000
    0x00, 0x2F,         // 伝文長 :47 byte
    0x00,               // ユニット ID :0
    0x10,               // FUNCTION CODE:0x10
    0x01, 0x04,         // レジスタアドレス:0x0104 (リモート I/O R-IN ~)
    0x00, 0x14,         // 書き込みレジスタ数 :20
    0x28,               // バイト数 :40
    0x00, 0x00,         // リモート I/O
    0x00, 0x00,         // 運転データ No の選択
    0x00, 0x00,         // 固定 I/O(IN)START OFF
    0x00, 0x00,         // ダイレクトデータ運転 運転方式
    0x00, 0x00,         // ダイレクトデータ運転 位置(下位)
    0x00, 0x00,         // ダイレクトデータ運転 位置(上位)
    0x00, 0x00,         // ダイレクトデータ運転 速度(下位)
    0x00, 0x00,         // ダイレクトデータ運転 速度(上位)
    0x00, 0x00,         // ダイレクトデータ運転 起動・変速レート(下位)
    0x00, 0x00,         // ダイレクトデータ運転 起動・変速レート(上位)
    0x00, 0x00,         // ダイレクトデータ運転 停止レート(下位)
    0x00, 0x00,         // ダイレクトデータ運転 停止レート(上位)
    0x00, 0x00,         // ダイレクトデータ運転 運転電流
    0x00, 0x00,         // ダイレクトデータ運転 転送先
    0x00, 0x00,         // 予約(reserved)
    0x00, 0x00,         // リードパラメータ ID
    0x00, 0x00,         // ライトリクエスト
    0x00, 0x00,         // ライトパラメータ ID
    0x00, 0x00,         // ライトデータ(下位)
    0x00, 0x00          // ライトデータ(上位)
};

/// <summary>
/// ダイレクトデータ運転を実行するクエリ
/// </summary>
public static byte[] frm_ExeOpe = new byte[]
{
    0x00, 0x00,         // プロトコル ID :0x0000
    0x00, 0x2F,         // 伝文長 :47 byte
    0x00,               // ユニット ID :0
    0x10,               // FUNCTION CODE:0x10
    0x01, 0x04,         // レジスタアドレス:0x0104 (リモート I/O R-IN ~)
    0x00, 0x14,         // 書き込みレジスタ数 :20
    0x28,               // バイト数:40
    0x00, 0x00,         // リモート I/O
    0x00, 0x00,         // 運転データ No の選択
    0x01, 0x00,         // 固定 I/O (IN) TRIG ON
    0x00, 0x02,         // ダイレクトデータ運転 運転方式                :2 相対位置決め(指令位置基準)
    0x21, 0x34,         // ダイレクトデータ運転 位置 (下位)             :8500 step
    0x00, 0x00,         // ダイレクトデータ運転 位置 (上位)
    0x05, 0xDC,         // ダイレクトデータ運転 速度 (下位)             :2000 Hz
    0x00, 0x00,         // ダイレクトデータ運転 速度 (上位)
    0x07, 0xD0,         // ダイレクトデータ運転 起動・変速レート (下位)  :1.5 kHz/s
    0x00, 0x00,         // ダイレクトデータ運転 起動・変速レート (上位)
    0x05, 0xDC,         // ダイレクトデータ運転 停止レート (下位)       :1.5 kHz/s
    0x00, 0x00,         // ダイレクトデータ運転 停止レート (上位)
    0x03, 0xE8,         // ダイレクトデータ運転 運転電流                :100.0 %
    0x00, 0x00,         // ダイレクトデータ運転 転送先
    0x00, 0x00,         // 予約(reserved)
    0x00, 0x00,         // リードパラメータ ID
    0x00, 0x00,         // ライトリクエスト
    0x00, 0x00,         // ライトパラメータ ID
    0x00, 0x00,         // ライトデータ(下位)
    0x00, 0x00          // ライトデータ(上位)
};

/// <summary>
/// ダイレクトデータ運転のトリガを OFF するクエリ
/// </summary>
public static byte[] frm_ExeOpe_TrgOFF = new byte[]
{
    0x00, 0x00,         // プロトコル ID :0x0000
    0x00, 0x2F,         // 伝文長 :47 byte
```

```
    0x00,               // ユニット ID :0
    0x10,               // FUNCTION CODE:0x10
    0x01, 0x04,         // レジスタアドレス:0x0104 (リモート I/O R-IN ~)
    0x00, 0x14,         // 書き込みレジスタ数 :20
    0x28,               // バイト数:40
    0x00, 0x00,         // リモート I/O
    0x00, 0x00,         // 運転データ No の選択
    0x00, 0x00,         // 固定 I/O (IN) TRIG OFF
    0x00, 0x02,         // ダイレクトデータ運転 運転方式                :2 相対位置決め(指令位置基準)
    0x21, 0x34,         // ダイレクトデータ運転 位置 (下位)             :8500 step
    0x00, 0x00,         // ダイレクトデータ運転 位置 (上位)
    0x05, 0xDC,         // ダイレクトデータ運転 速度 (下位)             :2000 Hz
    0x00, 0x00,         // ダイレクトデータ運転 速度 (上位)
    0x07, 0xD0,         // ダイレクトデータ運転 起動・変速レート (下位)  :1.5 kHz/s
    0x00, 0x00,         // ダイレクトデータ運転 起動・変速レート (上位)
    0x05, 0xDC,         // ダイレクトデータ運転 停止レート (下位)       :1.5 kHz/s
    0x00, 0x00,         // ダイレクトデータ運転 停止レート (上位)
    0x03, 0xE8,         // ダイレクトデータ運転 運転電流                :100.0 %
    0x00, 0x00,         // ダイレクトデータ運転 転送先
    0x00, 0x00,         // 予約(reserved)
    0x00, 0x00,         // リードパラメータ ID
    0x00, 0x00,         // ライトリクエスト
    0x00, 0x00,         // ライトパラメータ ID
    0x00, 0x00,         // ライトデータ(下位)
    0x00, 0x00          // ライトデータ(上位)
};


static void Main(string[] args)
{
    Modbus();
}

public static void Modbus()
{
    Console.WriteLine("*****************************************");
    Console.WriteLine("* Modbus TCP sample program for AZ series *");
    Console.WriteLine("* C#                                    *");
    Console.WriteLine("*                                       *");
    Console.WriteLine("*****************************************");

    Console.WriteLine("Driver IP Address : {0}", Address);
    Console.WriteLine("Driver Port : {0}", Port);

    // TCP-IP 通信準備
    IPAddress driver = IPAddress.Parse(Address);
    int port = Port;
    IPEndPoint ipe = new IPEndPoint(driver, port);

    uint counter = 0;
    bool loop_flg = true;
    byte[] frm;
    byte[] transaction;

    // 03h ファンクションのレスポンスバッファ: 9+レジスタ数(28)*2
    byte[] buffer_mon = new byte[65];

    // 10h ファンクションのレスポンスバッファ: 12
    byte[] buffer_write = new byte[12];

    // TCP-IP 通信を開始
    try
    {
        // TCP クライアントを使用
        using (var client = new TcpClient())
        {
            client.ConnectAsync(ipe);
            using (var stream = client.GetStream())
            {
                var readKey = "";
                while (loop_flg)
                {
                    // 'q'または'Q'が入力されるまでループ
                    Console.WriteLine(System.Environment.NewLine);
```

```
Console.WriteLine("1:Monitor");
Console.WriteLine("2:Set Operation Data");
Console.WriteLine("3:Operate");
Console.WriteLine("4:Execute Direct Data Operation");
Console.WriteLine("Q:Quit program");

readKey = Console.ReadLine().ToLower();

switch (readKey)
{
    // Program Stop
    case "q":
        loop_flg = false;
        break;

    // モーターの検出位置を読み出し、ターミナルに表示する
    case "1":

        // 送信用フレームを作成
        frm = new byte[frm_Mon.Length + 2];

        // トランザクション ID の配列
        transaction = BitConverter.GetBytes((short)counter).Reverse().ToArray();
        Buffer.BlockCopy(transaction, 0, frm, 0, transaction.Length);

        // 送信クエリをコピー
        Buffer.BlockCopy(frm_Mon, 0, frm, 2, frm_Mon.Length);

        // クエリ送信
        stream.Write(frm, 0, frm.Length);

        // クエリをコンソールに出力
        Console.WriteLine("TX : \t {0}", BitConverter.ToString(frm));

        // レスポンス取得
        stream.Read(buffer_mon, 0, buffer_mon.Length);


        // レスポンスをコンソールに出力
        Console.WriteLine("RX : \t {0}", BitConverter.ToString(buffer_mon));

        // 検出位置をコンソールに出力
        var remoteIO = new byte[4];
        Buffer.BlockCopy(buffer_mon, 17, remoteIO, 0, 4);

        // レスポンスをコンソールに出力
        Console.WriteLine("検出位置 : \t {0}", BitConverter.ToString(remoteIO));

        counter++;
        break;

    // 運転データ No.0 の位置、速度を書き換え
    case "2":
        // 運転データ No.0 の位置
        // 送信用フレームを作成
        frm = new byte[frm_WrOpeData_Pos.Length + 2];

        // トランザクション ID の配列
        transaction = BitConverter.GetBytes((short)counter).Reverse().ToArray();
        Buffer.BlockCopy(transaction, 0, frm, 0, transaction.Length);

        // 送信クエリをコピー
        Buffer.BlockCopy(frm_WrOpeData_Pos, 0, frm, 2, frm_WrOpeData_Pos.Length);

        // クエリ送信
        stream.Write(frm, 0, frm.Length);

        // クエリをコンソールに出力
        Console.WriteLine("TX : \t {0}", BitConverter.ToString(frm));

        // レスポンス取得
        stream.Read(buffer_write, 0, buffer_write.Length);

        // レスポンスをコンソールに出力
        Console.WriteLine("RX : \t {0}", BitConverter.ToString(buffer_write));
```

```
                                    // カウンタ+1
                                    counter++;

                                    // ライトリクエストを OFF する
                                    // 送信用フレームを作成
                                    frm = new byte[frm_WrOpeData_Pos_WrRqoff.Length + 2];

                                    // トランザクション ID の配列
                                    transaction = BitConverter.GetBytes((short)counter).Reverse().ToArray();
                                    Buffer.BlockCopy(transaction, 0, frm, 0, transaction.Length);

                                    // 送信クエリをコピー
                                    Buffer.BlockCopy(frm_WrOpeData_Pos_WrRqoff, 0, frm, 2,
frm_WrOpeData_Pos_WrRqoff.Length);

                                    // クエリ送信
                                    stream.Write(frm, 0, frm.Length);

                                    // クエリをコンソールに出力
                                    Console.WriteLine("TX : \t {0}", BitConverter.ToString(frm));

                                    // レスポンス取得
                                    stream.Read(buffer_write, 0, buffer_write.Length);

                                    // レスポンスをコンソールに出力
                                    Console.WriteLine("RX : \t {0}", BitConverter.ToString(buffer_write));

                                    // カウンタ+1
                                    counter++;


                                    // 運転データ No.0 の速度
                                    // 送信用フレームを作成
                                    frm = new byte[frm_WrOpeData_Spd.Length + 2];

                                    // トランザクション ID の配列
                                    transaction = BitConverter.GetBytes((short)counter).Reverse().ToArray();
                                    Buffer.BlockCopy(transaction, 0, frm, 0, transaction.Length);

                                    // 送信クエリをコピー
                                    Buffer.BlockCopy(frm_WrOpeData_Spd, 0, frm, 2, frm_WrOpeData_Spd.Length);

                                    // クエリ送信
                                    stream.Write(frm, 0, frm.Length);

                                    // クエリをコンソールに出力
                                    Console.WriteLine("TX : \t {0}", BitConverter.ToString(frm));

                                    // レスポンス取得
                                    stream.Read(buffer_write, 0, buffer_write.Length);

                                    // レスポンスをコンソールに出力
                                    Console.WriteLine("RX : \t {0}", BitConverter.ToString(buffer_write));

                                    // カウンタ+1
                                    counter++;

                                    // ライトリクエストを OFF する
                                    // 送信用フレームを作成
                                    frm = new byte[frm_WrOpeData_Spd_WrRqoff.Length + 2];

                                    // トランザクション ID の配列
                                    transaction = BitConverter.GetBytes((short)counter).Reverse().ToArray();
                                    Buffer.BlockCopy(transaction, 0, frm, 0, transaction.Length);

                                    // 送信クエリをコピー
                                    Buffer.BlockCopy(frm_WrOpeData_Spd_WrRqoff, 0, frm, 2,
frm_WrOpeData_Spd_WrRqoff.Length);

                                    // クエリ送信
                                    stream.Write(frm, 0, frm.Length);

                                    // クエリをコンソールに出力
                                    Console.WriteLine("TX : \t {0}", BitConverter.ToString(frm));
```

```
            // レスポンス取得
            stream.Read(buffer_write, 0, buffer_write.Length);

            // レスポンスをコンソールに出力
            Console.WriteLine("RX : \t {0}", BitConverter.ToString(buffer_write));

            // カウンタ+1
            counter++;
            break;

        // 固定 I/O の START を ON する
        case "3":
            // 固定 IO の START を ON するクエリ
            // 送信用フレームを作成
            frm = new byte[frm_FixedIn_Start.Length + 2];

            // トランザクション ID の配列
            transaction = BitConverter.GetBytes((short)counter).Reverse().ToArray();
            Buffer.BlockCopy(transaction, 0, frm, 0, transaction.Length);

            // 送信クエリをコピー
            Buffer.BlockCopy(frm_FixedIn_Start, 0, frm, 2, frm_FixedIn_Start.Length);

            // クエリ送信
            stream.Write(frm, 0, frm.Length);

            // クエリをコンソールに出力
            Console.WriteLine("TX : \t {0}", BitConverter.ToString(frm));

            // レスポンス取得
            stream.Read(buffer_write, 0, buffer_write.Length);

            // レスポンスをコンソールに出力
            Console.WriteLine("RX : \t {0}", BitConverter.ToString(buffer_write));

            counter++;

            // 固定 IO の START を OFF するクエリ
            // 送信用フレームを作成
            frm = new byte[frm_FixedIn_Stop.Length + 2];

            // トランザクション ID の配列
            transaction = BitConverter.GetBytes((short)counter).Reverse().ToArray();
            Buffer.BlockCopy(transaction, 0, frm, 0, transaction.Length);

            // 送信クエリをコピー
            Buffer.BlockCopy(frm_FixedIn_Stop, 0, frm, 2, frm_FixedIn_Stop.Length);

            // クエリ送信
            stream.Write(frm, 0, frm.Length);

            // クエリをコンソールに出力
            Console.WriteLine("TX : \t {0}", BitConverter.ToString(frm));

            // レスポンス取得
            stream.Read(buffer_write, 0, buffer_write.Length);

            // レスポンスをコンソールに出力
            Console.WriteLine("RX : \t {0}", BitConverter.ToString(buffer_write));

            counter++;
            break;

        // ダイレクトデータ運転を実行する
        case "4":
            // ダイレクトデータ運転を実行するクエリ
            // 送信用フレームを作成
            frm = new byte[frm_ExeOpe.Length + 2];

            // トランザクション ID の配列
            transaction = BitConverter.GetBytes((short)counter).Reverse().ToArray();
            Buffer.BlockCopy(transaction, 0, frm, 0, transaction.Length);

            // 送信クエリをコピー
```

```
                                    Buffer.BlockCopy(frm_ExeOpe, 0, frm, 2, frm_ExeOpe.Length);

                                    // クエリ送信
                                    stream.Write(frm, 0, frm.Length);

                                    // クエリをコンソールに出力
                                    Console.WriteLine("TX : \t {0}", BitConverter.ToString(frm));

                                    // レスポンス取得
                                    stream.Read(buffer_write, 0, buffer_write.Length);

                                    // レスポンスをコンソールに出力
                                    Console.WriteLine("RX : \t {0}", BitConverter.ToString(buffer_write));

                                    counter++;

                                    // ダイレクトデータ運転のトリガを OFF するクエリ
                                    // 送信用フレームを作成
                                    frm = new byte[frm_ExeOpe_TrgOFF.Length + 2];

                                    // トランザクション ID の配列
                                    transaction = BitConverter.GetBytes((short)counter).Reverse().ToArray();
                                    Buffer.BlockCopy(transaction, 0, frm, 0, transaction.Length);

                                    // 送信クエリをコピー
                                    Buffer.BlockCopy(frm_ExeOpe_TrgOFF, 0, frm, 2, frm_ExeOpe_TrgOFF.Length);

                                    // クエリ送信
                                    stream.Write(frm, 0, frm.Length);

                                    // クエリをコンソールに出力
                                    Console.WriteLine("TX : \t {0}", BitConverter.ToString(frm));

                                    // レスポンス取得
                                    stream.Read(buffer_write, 0, buffer_write.Length);

                                    // レスポンスをコンソールに出力
                                    Console.WriteLine("RX : \t {0}", BitConverter.ToString(buffer_write));

                                    counter++;
                                    break;

                                default:
                                    break;
                            }
                        }

                    }

                    // 切断
                    client.Close();
                }

            }
            catch (Exception ex)
            {
                Console.WriteLine("エラー: {0}", ex.ToString());
            }
        }

    }
}
```

オリエンタルモーター株式会社

作成：2023年 11月 17日