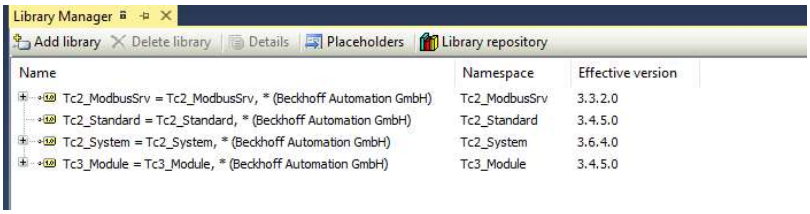
Read data from many Modbus TCP slaves



GVL

```
{attribute 'qualified_only'}

VAR_GLOBAL CONSTANT
    SLAVE_COUNT                 : WORD := 3; (* Number of Slaves which should be polled *)
    DATA_VOLUME_IN_WORD         : WORD:=  5; (* Number of WORDS which should be read out *)
END_VAR
```

MAIN

```
PROGRAM MAIN
VAR
    SlavePolling        :   SlavePolling;
    SlaveIpAddresses    :   ARRAY [1..GVL.SLAVE_COUNT] OF STRING(15) := ['192.168.1.100', '192.168.1.101', '192.168.1.102'];
    SlaveData           :   ARRAY [1..GVL.SLAVE_COUNT, 0..GVL.DATA_VOLUME_IN_WORD] OF WORD;
END_VAR
```

```
(*
This program reads out data from multiple Modbus TCP slaves
Number of MB slaves defined in GVL.SLAVE_COUNT
Number of registers to be read out from slaves defined in GVL.DATA_VOLUME_IN_WORD
SlaveIpAddresses : array with IP addresses of MB slaves
*)

SlavePolling(Start:= TRUE,
             SlaveIpAddresses:=SlaveIpAddresses ,
             RequestTime:= ,
             SlaveData=>SlaveData );
```

SlavePolling

```
FUNCTION_BLOCK SlavePolling
VAR_INPUT
    Start               :   BOOL;
    SlaveIpAddresses    :   ARRAY [1..GVL.SLAVE_COUNT] OF STRING(15);
    RequestTime         :   TIME := T#100MS;        (*Modbus Polling Time*)
END_VAR
VAR_OUTPUT
    SlaveData           :   ARRAY [1..GVL.SLAVE_COUNT, 0..GVL.DATA_VOLUME_IN_WORD] OF WORD;
END_VAR
VAR
    Index               :   UINT;
    DataIdx             :   WORD;
    tonRecycle          :   TON;
    MBReadRegs          :   FB_MBReadRegs;
    ErrorID             :   UDINT;
    PollCntSuccess      :   WORD;
    PollCntErrors       :   WORD;
    MBAddressRead       :   WORD:=1;                                    (*MB TCP adress to read out*)
    ReadRegs            :   ARRAY [0..GVL.DATA_VOLUME_IN_WORD] OF WORD;     (*MB TCP Number of registers to read*)

    SlaveCounter        :   WORD:=1;
END_VAR
```

SlavePolling*

```
tonRecycle(IN:= NOT tonRecycle.Q, PT:= RequestTime);

// Read Modbus TCP function 3
MBReadRegs(     sIPAddr         := SlaveIpAddresses[SlaveCounter],
                nTCPPort        := MODBUS_TCP_PORT,
                nUnitID         := 16#FF,
                nQuantity       := SIZEOF(ReadRegs)/2,
                nMBAddr         := MBAddressRead,
                cbLength        := SIZEOF(ReadRegs),
                pDestAddr       := ADR(ReadRegs));



CASE Index OF

    0:  // Idle

        ErrorID                 := 0;
        IF tonRecycle.Q AND Start THEN
            Index               := 10;
        END_IF

    10: // Execute Read Registers - Modbus function 3

        MBReadRegs.bExecute     := TRUE;
        Index                   := 20;


    20: // Read out one slave - wait for completion

        MBReadRegs.bExecute     := FALSE;
        IF NOT MBReadRegs.bBusy AND NOT MBReadRegs.bError THEN
                Index           := 30;
        END_IF;

        IF MBReadRegs.bError THEN
            ErrorID         := MBReadRegs.nErrId;
            PollCntErrors   := PollCntErrors + 1;

            FOR DataIdx:=0 TO GVL.DATA_VOLUME_IN_WORD DO
                ReadRegs[DataIdx] := 0;
            END_FOR

            FOR DataIdx:=0 TO GVL.DATA_VOLUME_IN_WORD DO
                SlaveData[SlaveCounter, DataIdx] := 0;
            END_FOR

            Index           := 40;
        END_IF

    30:
        // Save data from current polled slave into array
        FOR DataIdx:=0 TO GVL.DATA_VOLUME_IN_WORD DO
            SlaveData[SlaveCounter, DataIdx] := ReadRegs[DataIdx];
        END_FOR

        Index           := 40;


    40: // Polling of one slave completed. Select next slave IP address
        SlaveCounter := SlaveCounter + 1;
        IF SlaveCounter > GVL.SLAVE_COUNT THEN
            SlaveCounter := 1;
        END_IF

        Index := 0;

    100: // Error
        Index := 0;
END_CASE
```