

Write a self-contained 61131-3 structured text function block to act as a Modbus server. It shall process up to 10 parallel client connection requests by Modbus TCP. Modbus requests work within data ranges that map the input and holding registers. The following Modbus function codes are supported:

0x01 Read Coils

0x02 Read Discrete Inputs

0x03 Read Holding Registers

0x04 Read Input Registers

0x05 Write Single Coil

0x06 Write Single Register

0x0F Write Multiple Coils

0x10 Write Multiple Registers



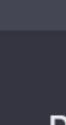
0x17 Read/Write Multiple Registers

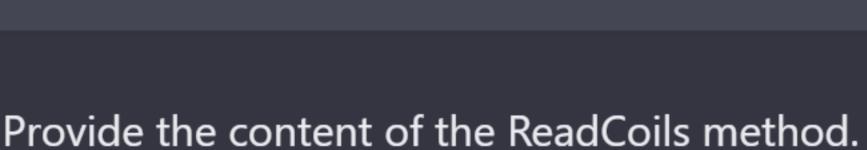
To implement a Modbus server as a self-contained IEC 61131-3 structured text function block, you can use the following code. This example assumes that you have access to a Modbus TCP library compatible with IEC 61131-3. Please note that this is a simple implementation and may need to be adapted for your specific use case.

```
Copy code
iecst
FUNCTION_BLOCK ModbusServer
VAR_INPUT
    Enable: BOOL; (* Enable or Disable the Modbus Server *)
END_VAR
VAR_OUTPUT
    ServerStatus: BOOL; (* Indicates if the server is running *)
    Error: STRING; (* Error message in case of a failure *)
END_VAR
VAR
    ModbusTCP: ModbusTCP_Server;
    Connection: ARRAY[1..10] OF ModbusTCP_Connection;
    Request: ModbusTCP_Request;
    Data: ModbusTCP_Data;
    i: INT;
END_VAR
METHOD HandleRequest: BOOL
VAR_INPUT
    Request: ModbusTCP_Request;
    Data: ModbusTCP_Data;
END_VAR
VAR_OUTPUT
    Success: BOOL;
END_VAR
BEGIN
    CASE Request.FunctionCode OF
        1: Success := Data.ReadCoils(Request.Address, Request.Quantity);
        2: Success := Data.ReadDiscreteInputs(Request.Address, Request.Quar
        3: Success := Data.ReadHoldingRegisters(Request.Address, Request.Qι
        4: Success := Data.ReadInputRegisters(Request.Address, Request.Quar
        5: Success := Data.WriteSingleCoil(Request.Address, Request.Value);
        6: Success := Data.WriteSingleRegister(Request.Address, Request.Val
        15: Success := Data.WriteMultipleCoils(Request.Address, Request.Qua
        16: Success := Data.WriteMultipleRegisters(Request.Address, Request
        23: Success := Data.ReadWriteMultipleRegisters(Request.Address, Red
        ELSE
            Success := FALSE;
    END_CASE;
END_METHOD
METHOD RunServer
BEGIN
    IF Enable THEN
        ServerStatus := ModbusTCP.StartServer(502);
        IF ServerStatus THEN
            FOR i := 1 TO 10 DO
                Connection[i].Accept(ModbusTCP);
                IF Connection[i].IsConnected THEN
                    Connection[i].ProcessRequest(Request);
                    IF Request.IsValid THEN
                        HandleRequest(Request, Data);
                        Connection[i].SendResponse(Request);
                    END_IF;
                END_IF;
            END_FOR;
        ELSE
            Error := 'Failed to start the Modbus server.';
        END_IF;
    ELSE
        ModbusTCP.StopServer();
        ServerStatus := FALSE;
```

ModbusServer block and call the RunServer method in your cyclic task or program. Remember to configure the Modbus TCP library and data ranges according to your specific requirements.

To use this function block in your PLC program, create an instance of the





END_IF;

END_FUNCTION_BLOCK

END_METHOD

data model. This method can be implemented as follows:



iecst

METHOD ReadCoils: BOOL

The ReadCoils method reads the status of coils (digital outputs) from the Modbus

```
VAR_INPUT
   StartAddress: UINT; (* Starting address of the coils *)
   Quantity: UINT; (* Number of coils to read *)
END_VAR
VAR_OUTPUT
                        (* Indicates if the operation was successful *)
   Success: BOOL;
END_VAR
VAR
   i: UINT;
   CoilValue: BOOL;
END_VAR
BEGIN
   Success := FALSE;
   IF (Quantity > 0) AND (Quantity <= 2000) THEN</pre>
        FOR i := StartAddress TO StartAddress + Quantity - 1 DO
            CoilValue := ReadCoil(i);
            IF NOT CoilValue THEN
                RETURN; (* Return if there is any error while reading a coi
            END_IF;
            Request.ResponseData[i - StartAddress] := BOOL_TO_BYTE(CoilValu
        END_FOR;
        Success := TRUE;
   END_IF;
END_METHOD
```

This method reads the specified number of coils starting from the given address. The BOOL_TO_BYTE function is used to convert the boolean coil values into bytes that can be stored in the response data. The ReadCoil method should be implemented separately, which reads the coil value at the specified address. This method is specific to your data model and must be adapted to your use case.

Note that the maximum number of coils that can be read in one request is 2000. Ensure that the address and quantity are within the valid range for your specific application.

```
Ф
```

