SGCWEBSOCKETS · TECHNICAL DOCUMENT

Modbus Server

TsgcModbusTCPServer and TsgcModbusRTUServer: Modbus TCP and Modbus RTU server for Delphi, C++Builder and .NET, answering from a thread-safe data model.

eSeGeCe

WWW.ESEGECE.COM

Overview

Turn a Delphi, C++Builder or .NET application into a Modbus device. Answer SCADA systems, HMIs and Modbus clients over TCP/IP or a serial line, from a thread-safe data model. The two servers answer from the same TsgcModbusDataModel: TsgcModbusTCPServer listens on TCP port 502, and TsgcModbusRTUServer is a Modbus RTU slave on a serial line. The server parses the requests, checks them and answers from the data model.

> **Availability.** Included in sgcWebSockets Enterprise (Delphi and .NET) and All-Access.

At a glance

COMPONENT CLASSES

`TsgcModbusTCPServer`, `TsgcModbusRTUServer`, `TsgcModbusDataModel`

STANDARDS / SPEC

Modbus Application Protocol V1.1b3

TRANSPORTS

TCP, RS-232, RS-485

PLATFORMS

TCP: every sgcWebSockets platform. RTU: Windows, Linux, macOS

FRAMEWORKS

VCL, FireMonkey, .NET

EDITION

Enterprise

Features

- **Function codes.** 01 to 08 (08 with sub-function 0), 15, 16, 22 (mask write, applied atomically), 23 (read/write multiple registers) and 43/14 (device identification, paged with More Follows).
- **Data model.** Coils, discrete inputs, holding registers and input registers, 10000 of each by default and up to 65536. Every Get and Set method is thread-safe, and a request beyond a table gets exception 02.

- **Read and write events.** OnReadRequest fires before the data is read, to load live values. OnWriteRequest fires before the data is written, to change the values or reject the request with a Modbus exception code.
- **Connections.** Each TCP connection is served on its own, and several requests in one packet are answered in order. A malformed header fires OnError and closes that connection. OnClientConnect and OnClientDisconnect report the peers.
- **Unit ids.** By default the TCP server answers every unit id. Set UnitId to answer one id only: the requests for other ids get no response and the connection stays open, as a gateway does with a missing device.
- **Device identification.** VendorName, ProductCode and MajorMinorRevision are the basic objects, and VendorUrl, ProductName and ModelName the regular ones, answered to function 43/14.

Technical specification

Standards & specs	Modbus Application Protocol Specification V1.1b3 · Modbus Messaging on TCP/IP Implementation Guide V1.0b · Modbus over Serial Line Specification V1.02
Component classes	`TsgcModbusTCPServer` and `TsgcModbusRTUServer` (unit `sgcModbus_Server`), `TsgcModbusDataModel` (unit `sgcModbus_DataModel`); in Delphi add `sgcIndustrial` to the uses clause. In .NET the classes have the same names in the namespace `esegece.sgcWebSockets`.
Function codes	01 to 08 (08 with sub-function 0), 15, 16, 22, 23 and 43/14
Data tables	Coils, discrete inputs, holding registers and input registers, 10000 of each by default and up to 65536
Default TCP port	502
RTU slave address	1 to 247; broadcast writes (address 0) are executed and never answered
Frameworks	VCL, FireMonkey, .NET
Platforms	TCP: every platform of sgcWebSockets. RTU: Windows, Linux and macOS

Main properties

The principal published properties used to configure and drive the component. Consult the online help for the full list.

`Active`	Starts or stops the server. The RTU server opens the serial port when it starts.
`DataModel`	The TsgcModbusDataModel the server answers from.
`Port`	TCP server: listening port, 502 by default.
`Bindings`	TCP server: the addresses and ports to listen on.
`UnitId`	TCP server: answer one unit id only (0 answers every id). RTU server: the slave address, 1 to 247.
`SerialPort`	RTU server: the TsgcSerialPort used as transport.

`DataModel.CoilCount`	Size of the coil table. DiscreteInputCount, HoldingRegisterCount and InputRegisterCount size the other three tables.
`DataModel.VendorName`	Basic identification object, with ProductCode and MajorMinorRevision.
`DataModel.VendorUrl`	Regular identification object, with ProductName and ModelName.

Main methods

The principal public methods exposed by the component.

`GetCoil()` / `SetCoil()`	Reads or writes one coil of the data model. GetCoils and SetCoils work on a range.
`GetDiscreteInput()` / `SetDiscreteInput()`	Reads or writes one discrete input. GetDiscreteInputs and SetDiscreteInputs work on a range.
`GetHoldingRegister()` / `SetHoldingRegister()`	Reads or writes one holding register. GetHoldingRegisters and SetHoldingRegisters work on a range.
`GetInputRegister()` / `SetInputRegister()`	Reads or writes one input register. GetInputRegisters and SetInputRegisters work on a range.
`ProcessRequest()`	Answers a Modbus request PDU from the data model.
`Assign()`	Copies the tables and the identification objects of another data model.

Events

The published events of the component.

`OnReadRequest`	Fires before the data is read, with the unit id, the function code, the address and the count. Load live values here.
`OnWriteRequest`	Fires before the data is written. Change the values, or set aExceptionCode to reject the request.
`OnError`	An error happened, for example a malformed header on a TCP connection.
`OnClientConnect`	TCP server: a client has connected.
`OnClientDisconnect`	TCP server: a client has disconnected.

Good to know

- The events run in the thread that serves the TCP connection, or in the frame thread of the RTU server. They are never synchronized with the main thread, because the server needs the answer of the handler to build the response. Do not access the user interface directly from them, and do not set Active to False from an event handler of the TCP server.
- To serve live values, write them into the data model whenever they change, or load them in OnReadRequest, which fires before every read with the function code, the address and the count. The data model methods can be called from the event, and from any other thread.
- To reject a write, set aExceptionCode in OnWriteRequest, for example to CS_MODBUS_EX_ILLEGAL_DATA_VALUE (03). The client receives that Modbus exception and the tables are left untouched.

Quick Start

Fill the data model, assign the events and set Active to True. The snippet that follows shows the typical **Start a Modbus TCP server** configuration, the same compiled sample as the component page.

> **About this scenario.** OnReadRequest refreshes a live value before it is read, and OnWriteRequest rejects setpoints out of range.

Delphi (VCL / FireMonkey)

```
uses
  sgcIndustrial;

procedure TForm1.StartServer;
begin
  FServer := TsgcModbusTCPServer.Create(Self);
  FServer.Port := 502;
  FServer.DataModel.HoldingRegisterCount := 1000;
  FServer.DataModel.SetHoldingRegister(0, 1234);
  FServer.DataModel.SetCoil(5, True);
  FServer.DataModel.VendorName := 'eSeGeCe';
  FServer.DataModel.ProductCode := 'PLC-1';
  FServer.DataModel.MajorMinorRevision := '1.0';
  FServer.OnReadRequest := OnModbusReadRequest;
  FServer.OnWriteRequest := OnModbusWriteRequest;
  FServer.Active := True;
end;

procedure TForm1.OnModbusReadRequest(Sender: TObject; aUnitId,
  aFunctionCode: Byte; aAddress, aCount: Word; var aExceptionCode: Byte);
begin
  // live value: refresh input register 0 before the client reads it
  if aFunctionCode = CS_MODBUS_FC_READ_INPUT_REGISTERS then
    FServer.DataModel.SetInputRegister(0, FTemperature);
end;

procedure TForm1.OnModbusWriteRequest(Sender: TObject; aUnitId,
  aFunctionCode: Byte; aAddress, aCount: Word; var aBits: TsgcModbusBits;
  var aRegisters: TsgcModbusRegisters; var aExceptionCode: Byte);
var
  i: Integer;
begin
  // connection thread: setpoints accept 0..1000 only
  if (aFunctionCode = CS_MODBUS_FC_WRITE_SINGLE_REGISTER) or
    (aFunctionCode = CS_MODBUS_FC_WRITE_MULTIPLE_REGISTERS) then
    for i := 0 to Length(aRegisters) - 1 do
      if aRegisters[i] > 1000 then
      begin
        aExceptionCode := CS_MODBUS_EX_ILLEGAL_DATA_VALUE;
        Exit;
      end;
end;
```

C++ Builder

```
#include "sgcIndustrial.hpp"

void __fastcall TForm1::StartServer()
{
  FServer = new TsgcModbusTCPServer(this);
  FServer->Port = 502;
  FServer->DataModel->HoldingRegisterCount = 1000;
  FServer->DataModel->SetHoldingRegister(0, 1234);
  FServer->DataModel->SetCoil(5, true);
  FServer->DataModel->VendorName = "eSeGeCe";
  FServer->DataModel->ProductCode = "PLC-1";
  FServer->DataModel->MajorMinorRevision = "1.0";
  FServer->OnReadRequest = OnModbusReadRequest;
  FServer->OnWriteRequest = OnModbusWriteRequest;
  FServer->Active = true;
}

void __fastcall TForm1::OnModbusReadRequest(TObject *Sender, Byte aUnitId,
  Byte aFunctionCode, Word aAddress, Word aCount, Byte &aExceptionCode)
{
  // live value: refresh input register 0 before the client reads it
  if (aFunctionCode == CS_MODBUS_FC_READ_INPUT_REGISTERS)
    FServer->DataModel->SetInputRegister(0, FTemperature);
}

void __fastcall TForm1::OnModbusWriteRequest(TObject *Sender, Byte aUnitId,
  Byte aFunctionCode, Word aAddress, Word aCount, TsgcModbusBits &aBits,
  TsgcModbusRegisters &aRegisters, Byte &aExceptionCode)
{
  // connection thread: setpoints accept 0..1000 only
  if ((aFunctionCode == CS_MODBUS_FC_WRITE_SINGLE_REGISTER) ||
      (aFunctionCode == CS_MODBUS_FC_WRITE_MULTIPLE_REGISTERS))
    for (int i = 0; i < aRegisters.Length; i++)
      if (aRegisters[i] > 1000)
      {
        aExceptionCode = CS_MODBUS_EX_ILLEGAL_DATA_VALUE;
        return;
      }
}
```

.NET (C#)

```
using System;
using esegece.sgcWebSockets;

class PlcServer
{
    TsgcModbusTCPServer FServer = new TsgcModbusTCPServer();
    ushort FTemperature = 215;

    public void StartServer()
    {
        FServer.Port = 502;
        FServer.DataModel.HoldingRegisterCount = 1000;
        FServer.DataModel.SetHoldingRegister(0, 1234);
        FServer.DataModel.SetCoil(5, true);
        FServer.DataModel.VendorName = "eSeGeCe";
        FServer.DataModel.ProductCode = "PLC-1";
        FServer.DataModel.MajorMinorRevision = "1.0";
        FServer.OnReadRequest += OnModbusReadRequest;
        FServer.OnWriteRequest += OnModbusWriteRequest;
        FServer.Active = true;
    }

    void OnModbusReadRequest(object Sender, byte aUnitId, byte aFunctionCode,
        ushort aAddress, ushort aCount, ref byte aExceptionCode)
    {
        // live value: refresh input register 0 before the client reads it
        if (aFunctionCode == sgcModbus_Classes.CS_MODBUS_FC_READ_INPUT_REGISTERS)
            FServer.DataModel.SetInputRegister(0, FTemperature);
    }

    void OnModbusWriteRequest(object Sender, byte aUnitId, byte aFunctionCode,
        ushort aAddress, ushort aCount, ref bool[] aBits, ref ushort[] aRegisters,
        ref byte aExceptionCode)
    {
        // connection thread: setpoints accept 0..1000 only
        if (aFunctionCode == sgcModbus_Classes.CS_MODBUS_FC_WRITE_SINGLE_REGISTER ||
            aFunctionCode == sgcModbus_Classes.CS_MODBUS_FC_WRITE_MULTIPLE_REGISTERS)
            foreach (ushort vValue in aRegisters)
                if (vValue > 1000)
                {
                    aExceptionCode = sgcModbus_Classes.CS_MODBUS_EX_ILLEGAL_DATA_VALUE;
                    return;
                }
    }
}
```

Common scenarios

The following scenarios use the same compiled samples as the component page. Each shows the configuration and method calls needed to drive the component through a specific real-world flow.

1 · Modbus RTU slave on RS-232 and RS-485

Link a TsgcSerialPort and set the slave address in UnitId (1 to 247). The server opens the port when it starts and answers from its data model.

Delphi (VCL / FireMonkey)

```
procedure TForm1.StartRTUServer;
begin
  FPort := TsgcSerialPort.Create(Self);
  FPort.Port := 'COM5';
  FPort.BaudRate := 19200;
  FPort.Parity := spEven;

  FRTUServer := TsgcModbusRTUServer.Create(Self);
  FRTUServer.SerialPort := FPort;
  FRTUServer.UnitId := 17;
  FRTUServer.DataModel.SetInputRegister(0, 215);
  // the server opens the port when it is closed
  FRTUServer.Active := True;
end;
```

C++ Builder

```
void __fastcall TForm1::StartRTUServer()
{
  FPort = new TsgcSerialPort(this);
  FPort->Port = "COM5";
  FPort->BaudRate = 19200;
  FPort->Parity = spEven;

  FRTUServer = new TsgcModbusRTUServer(this);
  FRTUServer->SerialPort = FPort;
  FRTUServer->UnitId = 17;
  FRTUServer->DataModel->SetInputRegister(0, 215);
  // the server opens the port when it is closed
  FRTUServer->Active = true;
}
```

.NET (C#)

```
using System;
using esegece.sgcWebSockets;

class RtuSlave
{
    TsgcSerialPort FPort = new TsgcSerialPort();
    TsgcModbusRTUServer FRTUServer = new TsgcModbusRTUServer();

    public void StartRTUServer()
    {
        FPort.Port = "COM5";
        FPort.BaudRate = 19200;
        FPort.Parity = TsgcSerialParity.spEven;

        FRTUServer.SerialPort = FPort;
        FRTUServer.UnitId = 17;
        FRTUServer.DataModel.SetInputRegister(0, 215);
        // the server opens the port when it is closed
        FRTUServer.Active = true;
    }
}
```

- Frames end at their expected length, or after 3.5 characters of silence
- Frames with a bad CRC or for another slave address are ignored, as the specification requires
- Broadcast writes (address 0) are executed and never answered
- The same data model, events and function codes as the TCP server

Sources used to build this document

Every external claim links back to a primary source. The online-help references decode the canonical deep-link the company maintains for this component.

Primary standard / spec: Modbus Application Protocol Specification V1.1b3	www.modbus.org/file/secure/modbusprotocolspecification.pdf
Primary standard / spec: Modbus Messaging on TCP/IP Implementation Guide V1.0b	www.modbus.org/file/secure/messagingimplementationguide.pdf
Primary standard / spec: Modbus over Serial Line Specification V1.02	www.modbus.org/file/secure/modbusoverserial.pdf
Online help: TsgcModbusTCPServer	www.esegece.com/help/sgcWebSockets/#t=Components%2FIndustrial%2FTsgcModbusTCPServer.htm
Online help: TsgcModbusRTUServer	www.esegece.com/help/sgcWebSockets/#t=Components%2FIndustrial%2FTsgcModbusRTUServer.htm
Online help: TsgcModbusDataModel	www.esegece.com/help/sgcWebSockets/#t=Components%2FIndustrial%2FTsgcModbusDataModel.htm
Demo project (in the sgcWebSockets package, Delphi, C++Builder and .NET)	`Industrial\03.Modbus_TCP_Server`
Component page	www.esegece.com/products/websockets/industrial/modbus-server/
Product page	www.esegece.com/products/websockets/

> **Document scope.** This document covers the publicly-documented surface of the Modbus TCP and RTU server components shipped with sgcWebSockets. For full property, method and event reference consult the online help linked above.