SGCWEBSOCKETS · TECHNICAL DOCUMENT

Modbus Client

TsgcModbusTCPClient and TsgcModbusRTUClient: Modbus TCP and Modbus RTU client for Delphi, C++Builder and .NET, with typed 32-bit values, retries, TLS and gateway support.

eSeGeCe

WWW.ESEGECE.COM

Overview

Read and write the coils and registers of PLCs, energy meters and drives from Delphi, C++Builder and .NET, over TCP/IP or over an RS-232 or RS-485 serial line. The two clients have the same methods: TsgcModbusTCPClient connects to a Modbus TCP device or gateway, and TsgcModbusRTUClient is a Modbus RTU master on a serial line. The framing, timing and error handling of Modbus are done by the component, so your code only deals with addresses and values.

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

At a glance

COMPONENT CLASSES

`TsgcModbusTCPClient`, `TsgcModbusRTUClient`

STANDARDS / SPEC

Modbus Application Protocol V1.1b3

TRANSPORTS

TCP, TLS, RS-232, RS-485

PLATFORMS

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

FRAMEWORKS

VCL, FireMonkey, .NET

EDITION

Enterprise

Features

- **Function codes.** 01 to 06, 15, 16, 22 (mask write), 23 (read/write multiple registers) and 43/14 (device identification, with paging). The RTU client adds 07 (read exception status) and 08 (diagnostics).
- **Typed values.** ReadInt32, ReadUInt32, ReadFloat32, ReadString and their write twins, plus reads from input registers. WordOrder selects ABCD, DCBA, BADC or CDAB to match the device.

- **Thread-safe requests.** Call the methods from any thread: the requests are serialized and every response is matched by its transaction id. Timeout applies to each attempt and Retries adds more attempts.
- **Clear errors.** A Modbus exception reply raises EsgcModbusException with the exception and function codes, no answer raises EsgcModbusTimeout, and every other problem raises EsgcModbusError. Quantities out of the specification are rejected before anything is sent.
- **TLS, proxy and reconnect.** The TCP client supports TLS, Proxy and WatchDog for automatic reconnection. A request that is waiting when the connection drops raises at once, without waiting for the timeout.
- **Gateways and unit ids.** Every method has an overload with a leading unit id, so one TCP connection reaches all the devices behind a Modbus TCP to RTU gateway without changing UnitId.

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	`TsgcModbusTCPClient` and `TsgcModbusRTUClient` (unit `sgcModbus_Client`; in Delphi add `sgcIndustrial` to the uses clause). In .NET the classes have the same names in the namespace `esegece.sgcWebSockets`.
Function codes	01, 02, 03, 04, 05, 06, 15, 16, 22, 23 and 43/14. RTU only: 07 and 08.
Word orders	`mwoABCD`, `mwoDCBA`, `mwoBADC`, `mwoCDAB`
Default TCP port	502 (`CS_MODBUS_TCP_DEFAULT_PORT`)
Exceptions	`EsgcModbusException` (exception reply), `EsgcModbusTimeout` (no answer), `EsgcModbusError` (any other problem)
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.

`UnitId`	Unit id (slave address) of the device the requests go to.
`Timeout`	Milliseconds each attempt waits for the response.
`Retries`	Additional attempts after a timeout.
`WordOrder`	Order of the two registers of a 32-bit value: ABCD, DCBA, BADC or CDAB.
`NotifyEvents`	How the events are delivered.
`Active`	TCP client: connects to or disconnects from the device.
`Host`	TCP client: address of the device or gateway.

`Port`	TCP client: TCP port, 502 by default.
`ConnectTimeout`	TCP client: connection timeout in milliseconds.
`TLS / TLSOptions`	TCP client: secure the connection with TLS.
`Proxy`	TCP client: connect through a proxy.
`WatchDog`	TCP client: reconnect automatically when the connection drops.
`SerialPort`	RTU client: the open TsgcSerialPort used as transport.
`TurnaroundDelay`	RTU client: pause after a broadcast write, in milliseconds.

Main methods

The principal public methods exposed by the component.

`ReadCoils()`	Function 01, reads coils.
`ReadDiscreteInputs()`	Function 02, reads discrete inputs.
`ReadHoldingRegisters()`	Function 03, reads holding registers.
`ReadInputRegisters()`	Function 04, reads input registers.
`WriteSingleCoil()`	Function 05, writes one coil.
`WriteSingleRegister()`	Function 06, writes one holding register.
`WriteMultipleCoils()`	Function 15, writes several coils.
`WriteMultipleRegisters()`	Function 16, writes several holding registers.
`MaskWriteRegister()`	Function 22, changes bits of a holding register with an AND and an OR mask.
`ReadWriteMultipleRegisters()`	Function 23, writes and reads holding registers in one transaction.
`ReadDeviceIdentification()`	Function 43/14, reads the identification objects of the device, with paging.
`ReadInt32() / WriteInt32()`	Signed 32-bit value in two holding registers, in the order set by WordOrder.
`ReadUInt32() / WriteUInt32()`	Unsigned 32-bit value in two holding registers.

`ReadFloat32() / WriteFloat32()`	IEEE 754 single precision value in two holding registers.
`ReadInputInt32() / ReadInputUInt32() / ReadInputFloat32()`	The same typed reads from input registers.
`ReadString() / WriteString()`	Text stored in holding registers.
`ReadExceptionStatus()`	RTU client, function 07.
`Diagnostics()`	RTU client, function 08.

Events

The published events of the component.

`OnConnect`	TCP client: the connection is established.
`OnDisconnect`	TCP client: the connection is closed.
`OnError`	TCP client: an error happened on the connection.

Good to know

- The methods take the protocol address, which starts at 0. The register that many manuals document as 40001 is holding register 0, so call ReadHoldingRegisters(0, 1).
- To read a 32-bit float, call ReadFloat32 with the address of the first of its two registers. If the value looks wrong, the device stores the bytes in another order: set WordOrder to the order the device uses. Many PLCs store the low word first, which is mwoCDAB.
- Each attempt waits Timeout milliseconds. After Retries + 1 attempts without a valid answer the request raises EsgcModbusTimeout, with the function code, the unit id and the number of attempts in the message.
- The RTU client does not open the serial port. Open the TsgcSerialPort yourself and leave its OnData event empty, because the client reads the replies with Read. A request raises EsgcModbusError when the port is not assigned, not open or has OnData assigned.

Quick Start

Set Host and UnitId, set Active to True and call the methods. The snippet that follows shows the typical **Connect, read and write over Modbus TCP** flow, the same compiled sample as the component page.

> **About this scenario.** Each request blocks until the matching response arrives, and raises an exception when the device reports an error or does not answer. The sample reads ten holding registers, writes a coil and reads a 32-bit float stored low word first.

Delphi (VCL / FireMonkey)

```
uses
  sgcIndustrial;

procedure TForm1.ConnectClient;
begin
  FClient := TsgcModbusTCPClient.Create(Self);
  FClient.Host := '192.168.1.50';
  FClient.Port := 502;
  FClient.UnitId := 1;
  FClient.Timeout := 1000;
  FClient.Retries := 2;
  FClient.OnConnect := OnModbusConnect;
  FClient.OnError := OnModbusError;
  FClient.Active := True;
end;

procedure TForm1.OnModbusConnect(Sender: TObject);
begin
  Memo1.Lines.Add('Connected');
end;

procedure TForm1.ReadValues;
var
  vRegisters: TsgcModbusRegisters;
  vTemperature: Single;
begin
  try
    vRegisters := FClient.ReadHoldingRegisters(0, 10);
    FClient.WriteSingleCoil(10, True);
    // 32-bit float stored low word first by many PLCs
    FClient.WordOrder := mwoCDAB;
    vTemperature := FClient.ReadFloat32(300);
    Memo1.Lines.Add(Format('HR0 = %d, temperature = %.1f',
      [vRegisters[0], vTemperature]));
  except
    on E: EsgcModbusException do
      Memo1.Lines.Add(Format('Device exception %d', [E.ExceptionCode]));
    on E: EsgcModbusTimeout do
      Memo1.Lines.Add('No response');
  end;
end;

procedure TForm1.OnModbusError(Sender: TObject; const aError: string);
begin
  Memo1.Lines.Add('Error: ' + aError);
end;
```

C++ Builder

```
#include "sgcIndustrial.hpp"

void __fastcall TForm1::ConnectClient()
{
  FClient = new TsgcModbusTCPClient(this);
  FClient->Host = "192.168.1.50"; // Port is 502 by default
  FClient->UnitId = 1;
  FClient->Timeout = 1000;
  FClient->Retries = 2;
  FClient->OnConnect = OnModbusConnect;
  FClient->OnError = OnModbusError;
  FClient->Active = true;
}

void __fastcall TForm1::OnModbusConnect(TObject *Sender)
{
  Memo1->Lines->Add("Connected");
}

void __fastcall TForm1::ReadValues()
{
  try
  {
    TsgcModbusRegisters vRegisters = FClient->ReadHoldingRegisters(0, 10);
    FClient->WriteSingleCoil(10, true);
    // 32-bit float stored low word first by many PLCs
    FClient->WordOrder = mwoCDAB;
    float vTemperature = FClient->ReadFloat32(300);
    Memo1->Lines->Add("HR0 = " + IntToStr(vRegisters[0]) + ", temperature = " +
      FloatToStrF(vTemperature, ffFixed, 7, 1));
  }
  catch (EsgcModbusException &E)
  {
    Memo1->Lines->Add("Device exception " + IntToStr(E.ExceptionCode));
  }
  catch (EsgcModbusTimeout &E)
  {
    Memo1->Lines->Add("No response");
  }
}

void __fastcall TForm1::OnModbusError(TObject *Sender, const String aError)
{
  Memo1->Lines->Add("Error: " + aError);
}
```

.NET (C#)

```
using System;
using esegece.sgcWebSockets;

class PlcClient
{
    TsgcModbusTCPClient FClient = new TsgcModbusTCPClient();

    public void ConnectClient()
    {
        FClient.Host = "192.168.1.50";
        FClient.Port = 502;
        FClient.UnitId = 1;
        FClient.Timeout = 1000;
        FClient.Retries = 2;
        FClient.OnConnect += OnModbusConnect;
        FClient.OnError += OnModbusError;
        FClient.Active = true;
    }

    void OnModbusConnect(object Sender, EventArgs e)
    {
        Console.WriteLine("Connected");
    }

    public void ReadValues()
    {
        try
        {
            ushort[] vRegisters = FClient.ReadHoldingRegisters(0, 10);
            FClient.WriteSingleCoil(10, true);
            // 32-bit float stored low word first by many PLCs
            FClient.WordOrder = TsgcModbusWordOrder.mwoCDAB;
            float vTemperature = FClient.ReadFloat32(300);
            Console.WriteLine($"HR0 = {vRegisters[0]}, temperature = {vTemperature:F1}");
        }
        catch (EsgcModbusException E)
        {
            Console.WriteLine("Device exception " + E.ExceptionCode);
        }
        catch (EsgcModbusTimeout)
        {
            Console.WriteLine("No response");
        }
    }

    void OnModbusError(object Sender, string aError)
    {
        Console.WriteLine("Error: " + aError);
    }
}
```

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 master on RS-232 and RS-485

Open a TsgcSerialPort, link it with SerialPort and set the slave address in UnitId. The client builds the RTU frame with its CRC16 and waits for the slave.

Delphi (VCL / FireMonkey)

```
procedure TForm1.ReadSlave;
var
  vRegisters: TsgcModbusRegisters;
begin
  FPort := TsgcSerialPort.Create(Self);
  FPort.Port := 'COM4';
  FPort.BaudRate := 19200;
  FPort.Parity := spEven;
  FPort.Open;

  FRTU := TsgcModbusRTUClient.Create(Self);
  FRTU.SerialPort := FPort;
  FRTU.UnitId := 17;
  FRTU.Timeout := 500;
  FRTU.Retries := 2;
  vRegisters := FRTU.ReadHoldingRegisters(107, 3);
  Memo1.Lines.Add(Format('Register 107 = %d', [vRegisters[0]]));

  // unit id 0 = broadcast: every slave executes the write, none answers
  FRTU.WriteSingleRegister(0, 1, 100);
end;
```

C++ Builder

```
void __fastcall TForm1::ReadSlave()
{
  FPort = new TsgcSerialPort(this);
  FPort->Port = "COM4";
  FPort->BaudRate = 19200;
  FPort->Parity = spEven;
  FPort->Open();

  FRTU = new TsgcModbusRTUClient(this);
  FRTU->SerialPort = FPort;
  FRTU->UnitId = 17;
  FRTU->Timeout = 500;
  FRTU->Retries = 2;
  TsgcModbusRegisters vRegisters = FRTU->ReadHoldingRegisters(107, 3);
  Memo1->Lines->Add("Register 107 = " + IntToStr(vRegisters[0]));

  // unit id 0 = broadcast: every slave executes the write, none answers
  FRTU->WriteSingleRegister(0, 1, 100);
}
```

.NET (C#)

```
using System;
using esegece.sgcWebSockets;

class RtuMaster
{
    TsgcSerialPort FPort = new TsgcSerialPort();
    TsgcModbusRTUClient FRTU = new TsgcModbusRTUClient();

    public void ReadSlave()
    {
        FPort.Port = "COM4";
        FPort.BaudRate = 19200;
        FPort.Parity = TsgcSerialParity.spEven;
        FPort.Open();

        FRTU.SerialPort = FPort;
        FRTU.UnitId = 17;
        FRTU.Timeout = 500;
        FRTU.Retries = 2;
        ushort[] vRegisters = FRTU.ReadHoldingRegisters(107, 3);
        Console.WriteLine("Register 107 = " + vRegisters[0]);

        // unit id 0 = broadcast: every slave executes the write, none answers
        FRTU.WriteSingleRegister(0, 1, 100);
    }
}
```

- Waits for 3.5 characters of silence before each request, as the serial line specification requires

- Reads the reply by its expected length, which works with USB adapters that deliver the bytes in bursts
- Checks the CRC of every reply, retries on a bad CRC and ignores the frames of other slaves
- Broadcast writes with unit id 0, followed by a TurnaroundDelay pause

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: TsgcModbusTCPClient	www.esegece.com/help/sgcWebSockets/#t=Components%2FIndustrial%2FTsgcModbusTCPClient.htm
Online help: TsgcModbusRTUClient	www.esegece.com/help/sgcWebSockets/#t=Components%2FIndustrial%2FTsgcModbusRTUClient.htm
Demo project (in the sgcWebSockets package, Delphi, C++Builder and .NET)	`Industrial\02.Modbus_TCP_Client`
Component page	www.esegece.com/products/websockets/industrial/modbus-client/
Product page	www.esegece.com/products/websockets/

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