SGCWEBSOCKETS · TECHNICAL DOCUMENT

Serial Port

TsgcSerialPort: RS-232 and RS-485 serial port for Delphi, C++Builder and .NET, with event driven or blocking reads on Windows, Linux and macOS.

eSeGeCe

WWW.ESEGECE.COM

Overview

TsgcSerialPort opens RS-232 and RS-485 ports and USB serial adapters from Delphi, C++Builder and .NET. It configures the line, writes data and receives it in an event or with blocking reads, on Windows, Linux and macOS. A background reader thread fills a 64 KB receive buffer, and the API is the same in Delphi, C++Builder and C#. The component is also the transport of the Modbus RTU client and server.

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

At a glance

COMPONENT CLASS
`TsgcSerialPort`

STANDARDS / SPEC
Win32 DCB, POSIX termios, Linux RS-485

TRANSPORTS
RS-232, RS-485, USB serial adapters

PLATFORMS
Windows, Linux, macOS (not Android or iOS)

FRAMEWORKS
VCL, FireMonkey, .NET

EDITION
Enterprise

Features

- **Line settings.** BaudRate, DataBits (5 to 8), Parity (none, odd, even, mark, space), StopBits (1, 1.5, 2) and FlowControl (RTS/CTS or XON/XOFF). Invalid combinations raise EsgcSerialError when the port opens.
- **Events or blocking reads.** With OnData assigned, every block of bytes goes to the event. Without it, the bytes wait in a 64 KB buffer and Read returns a whole reply: it waits for the first byte and ends after a silence you choose.
- **RS-485 direction control.** Set RtsControl to srcToggle for adapters that switch the direction with RTS. On Windows the driver raises RTS while it transmits, and on Linux the kernel RS-485 mode is

enabled. In .NET the toggle is timed by the component.

- **Ports and modem lines.** TsgcSerialPort.ListPorts lists the ports in natural order (COM2 before COM10). SetDTR, SetRTS and ModemStatus drive and read the modem lines.
- **Thread-safe.** Write can be called from any thread. Close works from any thread, also from an event handler, and wakes a blocked Read. In Delphi and C++Builder the events are queued to the main thread by default, and NotifyEvents changes that.
- **Clear errors.** Every failure raises EsgcSerialError, whose ErrorCode holds the operating system error code when the system reported one. A device unplugged while the port is open, or a full receive buffer, fires OnError.

Technical specification

Standards & specs	Windows DCB structure (serial line settings) · POSIX termios (Linux and macOS) · Linux RS-485 serial communications
Component class	`TsgcSerialPort` (unit `sgcSerial_Port`; in Delphi add `sgcIndustrial` to the uses clause). In .NET the class is `TsgcSerialPort` in the namespace `esegece.sgcWebSockets`.
Frameworks	VCL, FireMonkey, .NET
Platforms	In Delphi and C++Builder it works on Windows (Win32 and Win64) with the Win32 communications API, and on Linux64 and macOS with POSIX termios. It is not available on Android and iOS. The .NET version is built on System.IO.Ports.
Receive buffer	64 KB (`CS_SERIAL_BUFFER_SIZE`), filled by a background reader thread
Errors	`EsgcSerialError`, with `ErrorCode` holding the operating system error code

Main properties

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

`Active`	Opens or closes the port.
`Port`	Name of the port, such as `COM3` on Windows or `/dev/ttyUSB0` on Linux.
`BaudRate`	Line speed in bits per second.
`DataBits`	Data bits per character, 5 to 8.
`Parity`	`spNone`, `spOdd`, `spEven`, `spMark` or `spSpace`.
`StopBits`	`ssOne`, `ssOnePointFive` or `ssTwo`.
`FlowControl`	`sfcNone`, `sfcRtsCts`, `sfcDtrDsr` or `sfcXonXoff`.
`RtsControl`	`srcDisable`, `srcEnable`, `srcHandshake` or `srcToggle` (RS-485 direction control).

`ReadTimeout`	Read timeout in milliseconds.
`WriteTimeout`	Write timeout in milliseconds.
`NotifyEvents`	How the events are delivered. By default they are queued to the main thread.

Main methods

The principal public methods exposed by the component.

`Open()`	Opens the port with the current line settings.
`Close()`	Closes the port. Works from any thread, also from an event handler, and wakes a blocked Read.
`Write()`	Writes a byte array and returns the number of bytes written. Can be called from any thread.
`WriteString()`	Writes a string.
`Read()`	Blocking read: waits for the first byte and returns when the device has been silent for the given time. Only while OnData is not assigned.
`Purge()`	Discards the bytes waiting in the receive buffer.
`SetDTR()`	Raises or lowers the DTR line.
`SetRTS()`	Raises or lowers the RTS line.
`ModemStatus()`	Returns the state of the CTS, DSR, RI and DCD lines.
`BufferCount()`	Number of bytes waiting in the receive buffer.
`ListPorts()`	Class method that lists the serial ports of the system in natural order.

Events

The published events of the component.

`OnOpen`	The port has been opened.
`OnClose`	The port has been closed.
`OnData`	A block of bytes has been received.

`OnError`	An error happened while the port is open, for example the device was unplugged or the receive buffer is full.

Good to know

- Use OnData for devices that send data on their own, such as a barcode reader or a scale. Use Read for request and reply protocols: send the command, then read until the device stops sending. The two are exclusive: Read raises EsgcSerialError while OnData is assigned, and only one thread may call Read at the same time.
- An RS-485 adapter that switches the direction automatically needs no setting. An adapter that switches the direction with the RTS line needs RtsControl set to srcToggle. On Linux the component then enables the RS-485 mode of the driver and raises EsgcSerialError when the driver does not support it. macOS does not support srcToggle.

Quick Start

Set the port name and the line settings, assign the events and call Open. The snippet that follows shows the typical **Open a port and receive data** configuration, the same compiled sample as the component page.

> **About this scenario.** The device is on COM3 at 19200 baud, 8 data bits, even parity and one stop bit. The sample sends an AT command, and OnData receives every block of bytes as it arrives.

Delphi (VCL / FireMonkey)

```
uses
  sgcIndustrial;

procedure TForm1.OpenPort;
begin
  FPort := TsgcSerialPort.Create(Self);
  FPort.Port := 'COM3';
  FPort.BaudRate := 19200;
  FPort.DataBits := 8;
  FPort.Parity := spEven;
  FPort.StopBits := ssOne;
  FPort.OnData := OnSerialData;
  FPort.OnError := OnSerialError;
  FPort.Open;
  FPort.WriteString('AT' + #13);
end;

procedure TForm1.OnSerialData(Sender: TObject; const aData: TBytes);
begin
  // queued to the main thread by default (NotifyEvents = neAsynchronous)
  Memo1.Lines.Add(IntToStr(Length(aData)) + ' bytes received');
end;

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

C++ Builder

```
#include "sgcIndustrial.hpp"

void __fastcall TForm1::OpenPort()
{
  FPort = new TsgcSerialPort(this);
  FPort->Port = "COM3";
  FPort->BaudRate = 19200;
  FPort->DataBits = 8;
  FPort->Parity = spEven;
  FPort->StopBits = ssOne;
  FPort->OnData = OnSerialData;
  FPort->OnError = OnSerialError;
  FPort->Open();
  FPort->WriteString("AT\r");
}

void __fastcall TForm1::OnSerialData(TObject *Sender, const TBytes aData)
{
  // queued to the main thread by default (NotifyEvents = neAsynchronous)
  Memo1->Lines->Add(IntToStr(aData.Length) + " bytes received");
}

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

.NET (C#)

```
using System;
using esegece.sgcWebSockets;

class SerialTerminal
{
    TsgcSerialPort FPort = new TsgcSerialPort();

    public void OpenPort()
    {
        FPort.Port = "COM3";
        FPort.BaudRate = 19200;
        FPort.DataBits = 8;
        FPort.Parity = TsgcSerialParity.spEven;
        FPort.StopBits = TsgcSerialStopBits.ssOne;
        FPort.OnData += OnSerialData;
        FPort.OnError += OnSerialError;
        FPort.Open();
        FPort.WriteString("AT\r");
    }

    void OnSerialData(object Sender, byte[] aData)
    {
        Console.WriteLine(aData.Length + " bytes received");
    }

    void OnSerialError(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 · Request and reply

Leave OnData empty, write the command and call Read: it waits up to 1000 ms for the first byte and returns when the device has been silent for 50 ms.

Delphi (VCL / FireMonkey)

```
procedure TForm1.QueryDevice;
var
  vData: TBytes;
  vCount: Integer;
begin
  // OnData is not assigned: the bytes wait in the receive buffer for Read
  FPort.Purge;
  FPort.WriteString('*IDN?' + #10);
  // wait up to 1000 ms for the first byte, the reply ends after 50 ms of silence
  vCount := FPort.Read(vData, 256, 1000, 50);
  if vCount = 0 then
    Memo1.Lines.Add('No reply')
  else
    Memo1.Lines.Add(IntToStr(vCount) + ' bytes read');
end;
```

C++ Builder

```
void __fastcall TForm1::QueryDevice()
{
  // OnData is not assigned: the bytes wait in the receive buffer for Read
  FPort->Purge();
  FPort->WriteString("*IDN?\n");
  // wait up to 1000 ms for the first byte, the reply ends after 50 ms of silence
  TBytes vData;
  int vCount = FPort->Read(vData, 256, 1000, 50);
  if (vCount == 0)
    Memo1->Lines->Add("No reply");
  else
    Memo1->Lines->Add(IntToStr(vCount) + " bytes read");
}
```

.NET (C#)

```
using System;
using esegece.sgcWebSockets;

class SerialQuery
{
    TsgcSerialPort FPort = new TsgcSerialPort();

    public void QueryDevice()
    {
        // OnData is not assigned: the bytes wait in the receive buffer for Read
        FPort.Port = "COM3";
        FPort.Open();
        FPort.Purge();
        FPort.WriteString("*IDN?\n");
        // wait up to 1000 ms for the first byte, the reply ends after 50 ms of silence
        byte[] vData = FPort.Read(256, 1000, 50);
        if (vData.Length == 0)
            Console.WriteLine("No reply");
        else
            Console.WriteLine(vData.Length + " bytes read");
    }
}
```

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: Windows DCB structure (serial line settings)	learn.microsoft.com/en-us/windows/win32/api/winbase/ns-winbase-dcb
Primary standard / spec: POSIX termios (Linux and macOS)	man7.org/linux/man-pages/man3/termios.3.html
Primary standard / spec: Linux RS-485 serial communications	docs.kernel.org/driver-api/serial/serial-rs485.html
Online help: TsgcSerialPort	www.esegece.com/help/sgcWebSockets/#t=Components%2FIndustrial%2FTsgcSerialPort.htm
Demo project (in the sgcWebSockets package, Delphi, C++Builder and .NET)	`Industrial\01.Serial_Port`
Component page	www.esegece.com/products/websockets/industrial/serial-port/
Product page	www.esegece.com/products/websockets/

> **Document scope.** This document covers the publicly-documented surface of the serial port component shipped with sgcWebSockets. For full property, method and event reference consult the online help linked above.