SGCWEBSOCKETS · TECHNICAL DOCUMENT

OPC UA Client

TsgcOPCUAClient: OPC UA client for Delphi, C++Builder and .NET over binary opc.tcp, with signed and encrypted security policies, browse, read, write, method calls and subscriptions.

eSeGeCe

WWW.ESEGECE.COM

Overview

TsgcOPCUAClient connects Delphi, C++Builder and .NET applications to PLCs, SCADA systems and gateways that run an OPC UA server. It opens the secure channel and the session, browses the address space, reads and writes variables, calls methods and receives data changes and events through subscriptions. The secure channel, the session and the publish loop are handled by the component, so your code works with node ids and values.

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

At a glance

COMPONENT CLASS `TsgcOPCUAClient`	STANDARDS / SPEC OPC 10000-4, 10000-6, 10000-7
TRANSPORT OPC UA binary over opc.tcp	SECURITY None, Sign, SignAndEncrypt
FRAMEWORKS VCL, FireMonkey, .NET	EDITION Enterprise

Features

- **Services.** GetEndpoints, Browse and BrowseNext (continuation points are followed for you), TranslateBrowsePath, Read and Write of any attribute, and Call for methods. ReadValue and WriteValue take a node id string and use the data type of the node.
- **Security policies.** Basic256Sha256, Aes128_Sha256_RsaOaep and Aes256_Sha256_RsaPss, plus the deprecated Basic128Rsa15 and Basic256 for older servers, in Sign or SignAndEncrypt mode. The client checks the server signature and nonce and rejects an endpoint list that was changed on the way.

- **User identity.** Anonymous, user name and password (encrypted with the server key when the token policy asks for it) or an X.509 user certificate.
- **Certificates.** The application instance certificate is loaded from DER or PEM files, created on the first connect and saved to those files, or created in memory. A server certificate is rejected unless OnValidateServerCertificate accepts it.
- **Subscriptions.** Data change items with sampling interval, queue size, absolute or percent deadband and trigger; event items with select clauses and an optional where clause. Acknowledgements and the republishing of missing notifications are done by the client.
- **Automatic reconnect.** WatchDog reactivates the same session and republishes what was missed, or transfers or recreates the subscriptions in a new session. It works the same in Delphi, C++Builder and .NET.

Technical specification

Standards & specs	OPC 10000-2: Security Model · OPC 10000-4: Services · OPC 10000-6: Mappings (UA TCP, UA Secure Conversation, UA Binary) · OPC 10000-7: Profiles (security policies)
Component class	`TsgcOPCUAClient` (unit `sgcOPCUA_Client`; in Delphi add `sgcIndustrial` to the uses clause). In .NET the class is `TsgcOPCUAClient` in the namespace `esegece.sgcWebSockets`, package `esegece.sgcIndustrial`.
Frameworks	VCL, FireMonkey, .NET
Platforms	Delphi 7 to Delphi 13 and C++Builder on Windows (Win32 and Win64) and Linux64. .NET 8, .NET 9, .NET 10 and .NET Standard 2.0.
Not included	An OPC UA server, the HTTPS, JSON and WebSocket transports, HistoryRead, PubSub, Reverse Connect and the Global Discovery Server.
Errors	`EsgcOPCUAError`, with `StatusCode` holding the OPC UA status code

Main properties

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

`Active`	Connects or disconnects the client.
`EndpointUrl`	opc.tcp address of the server, such as `opc.tcp://192.168.1.60:4840`.
`SecurityPolicy`	`opcspNone`, `opcspBasic128Rsa15`, `opcspBasic256`, `opcspBasic256Sha256`, `opcspAes128_Sha256_RsaOaep` or `opcspAes256_Sha256_RsaPss`.
`SecurityMode`	`opcsmNone`, `opcsmSign` or `opcsmSignAndEncrypt`.
`Application`	Application name, URIs and the instance certificate files; `AutoGenerate` creates the certificate when it is missing.
`Authentication`	Identity type (`opcatAnonymous`, `opcatUserName`, `opcatCertificate`), user name and password, or user certificate

	files.
`Timeout`	Milliseconds for each connect step and each service call.
`SessionName`	Name of the session shown by the server.
`SessionTimeout`	Requested session timeout in milliseconds.
`AutoAcceptServerCertificate`	Accepts any server certificate. For test benches only; the default is False.
`WatchDog`	Automatic reconnect (`Enabled` , `Interval` in seconds, `Attempts`).
`NotifyEvents`	How the events are delivered. By default they are queued to the main thread.
`State`	Read only: disconnected, connecting, connected, reconnecting and the steps in between.

Main methods

The principal public methods exposed by the component.

`Connect()`	Selects the endpoint, opens the secure channel, creates and activates the session.
`Disconnect()`	Closes the session, deletes its subscriptions and closes the channel.
`GetEndpoints()`	Lists the endpoints of the server without opening a session.
`Browse()` / `BrowseNext()`	Lists the references of a node; the simple overload follows the continuation points.
`TranslateBrowsePath()`	Resolves a browse path such as `/2:Line1/2:Speed` to a node id.
`Read()` / `ReadValue()`	Reads attributes of several nodes, or the value of one node.
`Write()` / `WriteValue()`	Writes attributes of several nodes, or the value of one node with the data type of the node.
`Call()`	Calls a method of an object with its input arguments and returns the output arguments.
`CreateSubscription()` / `DeleteSubscription()`	Creates or deletes a subscription with its publishing interval and counts.

`SetPublishingMode()`	Pauses or resumes the notifications of a subscription.
`AddMonitoredItem()` / `AddEventMonitoredItem()`	Monitors the value of a node, or the events of a node with select and where clauses.
`ModifyMonitoredItem()` / `RemoveMonitoredItem()`	Changes the sampling, queue and filter of an item, or removes it.

Events

The published events of the component.

`OnConnect`	The session is active.
`OnDisconnect`	The client has disconnected or the connection was lost.
`OnError`	An error happened, with its OPC UA status code.
`OnStatusChange`	The state of the client changed.
`OnValidateServerCertificate`	Decide whether the server certificate is trusted.
`OnDataChange`	A monitored value changed.
`OnEvent`	An event arrived, with the selected fields.
`OnKeepAlive`	The server sent a keep-alive for a subscription with no notifications.
`OnReconnect`	WatchDog restored the connection, with the path taken.

Good to know

- Service calls block until the server answers or Timeout expires. With NotifyEvents set to neNoSync, the events run in the client threads: do not call Connect or other blocking services from them, and let WatchDog reconnect.
- Subscriptions and monitored items are owned by the client. With WatchDog they survive a reconnect: server ids can change, the ClientHandle of an item never does. Without WatchDog, create them again after Connect.
- Deadbands apply to numeric values only; the server rejects a deadband on other types with BadFilterNotAllowed.

Quick Start

Set the endpoint, choose the security policy and mode and call Connect. The snippet that follows shows the typical **Connect, read and write** configuration, the same compiled sample as the component page.

> **About this scenario.** The server listens on opc.tcp://192.168.1.60:4840 and accepts the None policy. The sample reads a temperature and writes a setpoint, with the data type taken from the node.

Delphi (VCL / FireMonkey)

```
uses
  sgcIndustrial;

procedure TForm1.ConnectAndRead;
begin
  FClient := TsgcOPCUAClient.Create(Self);
  FClient.EndpointUrl := 'opc.tcp://192.168.1.60:4840';
  FClient.SecurityPolicy := opcspNone;
  FClient.SecurityMode := opcsmNone;
  FClient.OnError := OnOPCUAError;
  FClient.Connect;
  // read the Value attribute of a variable
  Memo1.Lines.Add('Temperature: ' + VarToStr(FClient.ReadValue('ns=2;s=Line1.Temperature')));
  // the data type of the node is read first and used for the written value
  FClient.WriteValue('ns=2;s=Line1.Setpoint', 75.5);
end;

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

C++ Builder

```
#include "sgcIndustrial.hpp"

void __fastcall TForm1::ConnectAndRead()
{
  FClient = new TsgcOPCUAClient(this);
  FClient->EndpointUrl = "opc.tcp://192.168.1.60:4840";
  FClient->SecurityPolicy = opcspNone;
  FClient->SecurityMode = opcsmNone;
  FClient->OnError = OnOPCUAError;
  FClient->Connect();
  // read the Value attribute of a variable
  Memo1->Lines->Add("Temperature: " + VarToStr(FClient->ReadValue("ns=2;s=Line1.Temperature")));
  // the data type of the node is read first and used for the written value
  FClient->WriteValue("ns=2;s=Line1.Setpoint", 75.5);
}

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

.NET (C#)

```
using System;
using esegece.sgcWebSockets;

class PlantReader
{
    TsgcOPCUAClient FClient = new TsgcOPCUAClient();

    public void ConnectAndRead()
    {
        FClient.EndpointUrl = "opc.tcp://192.168.1.60:4840";
        FClient.SecurityPolicy = TsgcOPCUASecurityPolicy.opcspNone;
        FClient.SecurityMode = TsgcOPCUAMessageSecurityMode.opcsmNone;
        FClient.OnError += OnOPCUAError;
        FClient.Connect();
        // read the Value attribute of a variable
        Console.WriteLine("Temperature: " + FClient.ReadValue("ns=2;s=Line1.Temperature"));
        // the data type of the node is read first and used for the written value
        FClient.WriteValue("ns=2;s=Line1.Setpoint", 75.5);
    }

    void OnOPCUAError(object Sender, uint aStatusCode, 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 · Subscribe to data changes

Create a subscription with a publishing interval of 500 ms and monitor two variables, one of them with an absolute deadband of 0.5.

Delphi (VCL / FireMonkey)

```
procedure TForm1.Subscribe;
var
  oSubscription: TsgcOPCUASubscription;
begin
  FClient.OnDataChange := OnOPCUADataChange;
  // publishing interval 500 ms
  oSubscription := FClient.CreateSubscription(500);
  // sampling 250 ms, queue of 1, report changes larger than 0.5 only
  FClient.AddMonitoredItem(oSubscription, 'ns=2;s=Line1.Temperature', 250, 1, True,
    uadbtAbsolute, 0.5);
  FClient.AddMonitoredItem(oSubscription, 'ns=2;s=Line1.Pressure');
end;

procedure TForm1.OnOPCUADataChange(Sender: TObject; aSubscription: TsgcOPCUASubscription;
  aMonitoredItem: TsgcOPCUAMonitoredItem; const aValue: TsgcOPCUADataValue);
begin
  if (aValue.StatusCode and $80000000) = 0 then
    Memo1.Lines.Add('Item ' + IntToStr(aMonitoredItem.ClientHandle) + ': ' +
      VarToStr(aValue.Value.Value));
end;
```

C++ Builder

```
void __fastcall TForm1::Subscribe()
{
  FClient->OnDataChange = OnOPCUADataChange;
  // publishing interval 500 ms
  TsgcOPCUASubscription *oSubscription = FClient->CreateSubscription(500);
  // sampling 250 ms, queue of 1, report changes larger than 0.5 only
  FClient->AddMonitoredItem(oSubscription, "ns=2;s=Line1.Temperature", 250, 1, true,
    uadbtAbsolute, 0.5);
  FClient->AddMonitoredItem(oSubscription, "ns=2;s=Line1.Pressure");
}

void __fastcall TForm1::OnOPCUADataChange(TObject *Sender, TsgcOPCUASubscription *aSubscription,
  TsgcOPCUAMonitoredItem *aMonitoredItem, const TsgcOPCUADataValue &aValue)
{
  if ((aValue.StatusCode & 0x80000000) == 0)
    Memo1->Lines->Add("Item " + IntToStr((int)aMonitoredItem->ClientHandle) + ": " +
      VarToStr(aValue.Value.Value));
}
```

.NET (C#)

```
using System;
using esegece.sgcWebSockets;

class PlantMonitor
{
    public void Subscribe(TsgcOPCUAClient aClient)
    {
        aClient.OnDataChange += OnOPCUADataChange;
        // publishing interval 500 ms
        TsgcOPCUASubscription oSubscription = aClient.CreateSubscription(500);
        // sampling 250 ms, queue of 1, report changes larger than 0.5 only
        aClient.AddMonitoredItem(oSubscription, "ns=2;s=Line1.Temperature", 250, 1, true,
            TsgcOPCUADeadbandType.uadbtAbsolute, 0.5);
        aClient.AddMonitoredItem(oSubscription, "ns=2;s=Line1.Pressure");
    }

    void OnOPCUADataChange(object Sender, TsgcOPCUASubscription aSubscription,
        TsgcOPCUAMonitoredItem aMonitoredItem, TsgcOPCUADataValue aValue)
    {
        if ((aValue.StatusCode & 0x80000000) == 0)
            Console.WriteLine("Item " + aMonitoredItem.ClientHandle + ": " + aValue.Value.Value)
    }
}
```

2 · Signed and encrypted connection

Basic256Sha256 with SignAndEncrypt, an application certificate saved on the first run, a user name identity and a server certificate check.

Delphi (VCL / FireMonkey)

```
uses
  sgcIndustrial; // Delphi 7 to 2006: add sgcBase_Helpers (TBytes)

procedure TForm1.ConnectSecure;
begin
  FClient := TsgcOPCUAClient.Create(Self);
  FClient.EndpointUrl := 'opc.tcp://192.168.1.60:4840';
  FClient.SecurityPolicy := opcspBasic256Sha256;
  FClient.SecurityMode := opcsmSignAndEncrypt;
  // application instance certificate: created and saved on the first run
  FClient.Application.ApplicationName := 'Line1 Monitor';
  FClient.Application.CertificateFile := 'C:\pki\own\client.der';
  FClient.Application.PrivateKeyFile := 'C:\pki\own\client.pem';
  // user identity
  FClient.Authentication.AuthenticationType := opcatUserName;
  FClient.Authentication.User := 'operator';
  FClient.Authentication.Password := 'secret';
  // decide which server certificates you trust
  FClient.OnValidateServerCertificate := OnOPCUAValidateServerCertificate;
  FClient.Connect;
end;

procedure TForm1.OnOPCUAValidateServerCertificate(Sender: TObject;
  const aCertificate: TBytes; var aAccept: Boolean);
begin
  // FTrusted holds the DER bytes of the server certificate you trust
  aAccept := (Length(aCertificate) > 0) and (Length(aCertificate) = Length(FTrusted)) and
    CompareMem(@aCertificate[0], @FTrusted[0], Length(aCertificate));
end;
```

C++ Builder

```
void __fastcall TForm1::ConnectSecure()
{
  FClient = new TsgcOPCUAClient(this);
  FClient->EndpointUrl = "opc.tcp://192.168.1.60:4840";
  FClient->SecurityPolicy = opcspBasic256Sha256;
  FClient->SecurityMode = opcsmSignAndEncrypt;
  // application instance certificate: created and saved on the first run
  FClient->Application->ApplicationName = "Line1 Monitor";
  FClient->Application->CertificateFile = "C:\\pki\\own\\client.der";
  FClient->Application->PrivateKeyFile = "C:\\pki\\own\\client.pem";
  // user identity
  FClient->Authentication->AuthenticationType = opcatUserName;
  FClient->Authentication->User = "operator";
  FClient->Authentication->Password = "secret";
  // decide which server certificates you trust
  FClient->OnValidateServerCertificate = OnOPCUAValidateServerCertificate;
  FClient->Connect();
}

void __fastcall TForm1::OnOPCUAValidateServerCertificate(TObject *Sender,
  const TBytes aCertificate, bool &aAccept)
{
  // FTrusted holds the DER bytes of the server certificate you trust
  aAccept = (aCertificate.Length > 0) && (aCertificate.Length == FTrusted.Length) &&
    CompareMem((void*)&aCertificate[0], &FTrusted[0], aCertificate.Length);
}
```

.NET (C#)

```
using System.IO;
using System.Linq;
using esegece.sgcWebSockets;

class SecureClient
{
    TsgcOPCUAClient FClient = new TsgcOPCUAClient();
    // DER bytes of the server certificate you trust
    byte[] FTrusted = File.ReadAllBytes(@"C:\pki\trusted\server.der");

    public void ConnectSecure()
    {
        FClient.EndpointUrl = "opc.tcp://192.168.1.60:4840";
        FClient.SecurityPolicy = TsgcOPCUASecurityPolicy.opcspBasic256Sha256;
        FClient.SecurityMode = TsgcOPCUAMessageSecurityMode.opcsmSignAndEncrypt;
        // application instance certificate: created and saved on the first run
        FClient.Application.ApplicationName = "Line1 Monitor";
        FClient.Application.CertificateFile = @"C:\pki\own\client.der";
        FClient.Application.PrivateKeyFile = @"C:\pki\own\client.pem";
        // user identity
        FClient.Authentication.AuthenticationType = TsgcOPCUAAuthenticationType.opcatUserName;
        FClient.Authentication.User = "operator";
        FClient.Authentication.Password = "secret";
        // decide which server certificates you trust
        FClient.OnValidateServerCertificate += OnOPCUAValidateServerCertificate;
        FClient.Connect();
    }

    void OnOPCUAValidateServerCertificate(object Sender, byte[] aCertificate, ref bool aAccept)
    {
        aAccept = aCertificate ≠ null && aCertificate.SequenceEqual(FTrusted);
    }
}
```

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: OPC 10000-2: Security Model	reference.opcfoundation.org/Core/Part2/v105/docs/
Primary standard / spec: OPC 10000-4: Services	reference.opcfoundation.org/Core/Part4/v105/docs/
Primary standard / spec: OPC 10000-6: Mappings (UA TCP, UA Secure Conversation, UA Binary)	reference.opcfoundation.org/Core/Part6/v105/docs/
Primary standard / spec: OPC 10000-7: Profiles (security policies)	reference.opcfoundation.org/Core/Part7/v105/docs/
Online help: TsgcOPCUAClient	www.esegece.com/help/sgcWebSockets/#t=Components%2FIndustrial%2FTsgcOPCUAClient.htm
Online help: OPC UA Subscriptions	www.esegece.com/help/sgcWebSockets/#t=Components%2FIndustrial%2FOPC_UA_Subscriptions.htm
Demo project (in the sgcWebSockets package, Delphi, C++Builder and .NET)	`Industrial\05.OPCUA_Client`
Component page	www.esegece.com/products/websockets/industrial/opcua-client/
Product page	www.esegece.com/products/websockets/

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