SGCWEBSOCKETS · TECHNICAL DOCUMENT

WebSocket Client (Delphi)

The TsgcWebSocketClient component — RFC 6455 WebSocket client for Delphi VCL, FireMonkey, Lazarus and FPC over TCP and TLS.

eSeGeCe

WWW.ESEGECE.COM

Overview

Client WebSocket component — connects to any RFC 6455 WebSocket server and sends/receives text and binary messages.

At a glance

COMPONENT CLASS

`TsgcWebSocketClient`

STANDARDS / SPEC

WebSocket Protocol — RFC 6455

TRANSPORTS

TCP, TLS

PLATFORMS

Windows, macOS, Linux, iOS, Android

FRAMEWORKS

VCL, FireMonkey, Lazarus / FPC

EDITION

Standard / Professional / Enterprise

Features

- Native Delphi implementation with full ANSI/Unicode support.

Technical specification

Standards & specs	WebSocket Protocol — RFC 6455 · Per-message Deflate — RFC 7692
Component class	`TsgcWebSocketClient` (unit `sgcWebSocket_Client`)
Frameworks	VCL, FireMonkey, Lazarus / FPC
Platforms	Windows, macOS, Linux, iOS, Android

Main properties

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

`URL`	Write-only shortcut that populates Host, Port, TLS and parameters from a single WebSocket URL.
`Authentication`	Configures the credentials and scheme used to authenticate the WebSocket handshake.
`TLSOptions`	Configures certificates, TLS version, ALPN, IOHandler and other secure-connection details used when TLS is enabled.
`Active`	Opens or closes the WebSocket connection to the configured server.
`Host`	IP address or DNS name of the WebSocket server the client will connect to.
`Port`	TCP port used to connect to the WebSocket server.
`TLS`	Enables a secure TLS/SSL connection to the WebSocket server.
`HeartBeat`	Sends periodic WebSocket ping frames to keep the connection alive.
`WatchDog`	Automatically reconnects to the server after an unexpected disconnection.
`Proxy`	Routes the WebSocket connection through an HTTP or SOCKS proxy server.

Main methods

The principal public methods exposed by the component.

`WriteAndWaitData()`	Sends a text message and blocks the caller until the server responds with a text message or the timeout elapses.
`Start()`	Connects to the server asynchronously from a secondary thread so the calling thread is not blocked.
`Stop()`	Disconnects from the server asynchronously from a secondary thread so the calling thread is not blocked.
`Connect()`	Opens the WebSocket connection synchronously and blocks the caller until the handshake completes or the timeout elapses.
`Disconnect()`	Closes the WebSocket connection synchronously and blocks the caller until the disconnection completes or the timeout elapses.
`Connected()`	Queries the underlying socket to determine whether the client is currently connected to the server.
`Ping()`	Sends a WebSocket ping frame to the server and returns immediately without waiting for the pong response.
`WriteData()`	Sends a text message to the WebSocket server, optionally splitting it into fragments of a given size.

Public events

The component exposes the following published events; consult the online help for full event-handler signatures.

`OnBeforeConnect`	property OnBeforeConnect: TsgcWSOnBeforeConnectEvent; // TsgcWSOnBeforeConnectEvent = procedure(Sender: TObject) of object __property TsgcWSOnBeforeConnectEvent OnBeforeConnect; // typedef void __fast...
`OnBeforeHeartBeat`	Fires before each HeartBeat ping so the application can send a custom keep-alive message.
`OnBeforeWatchDog`	Fires before each WatchDog reconnection attempt so the application can adjust the target server or suppress the retry.
`OnBinary`	Fires every time the server sends a binary message to the client.
`OnConnect`	Fires when a WebSocket connection to the server has been successfully established.

`OnDisconnect`	Fires when the WebSocket connection has been dropped or closed.
`OnError`	Fires every time a WebSocket protocol error occurs on the connection.
`OnException`	Fires whenever an unhandled exception is raised while processing the connection.
`OnFragmented`	Fires for every fragment received when Options.FragmentedMessages is frgAll or frgOnlyFragmented.
`OnHandshake`	Fires when the client HTTP handshake is being built so custom headers can be added.
`OnLoadBalancerError`	Fires when LoadBalancer is enabled and an error occurs while communicating with the Load Balancer Server.
`OnMessage`	Fires every time the server sends a text message to the client.
`OnSChannelVerifyPeer`	Fires when SChannel is the TLS IOHandler so the application can verify the server certificate.
`OnSSLAfterCreateHandler`	Fires after the SSL handler has been created so its properties can be customized.
`OnSSLGetHandler`	Fires before the SSL handler is created so a custom handler instance can be supplied.
`OnSSLVerifyPeer`	Fires when VerifyCertificate is enabled so the application can accept or reject the server certificate.

Quick Start

Drop the component on a form, configure the properties below and activate it. The snippet that follows shows the typical **TsgcWebSocketClient | Client Exceptions** configuration sourced from the online help.

> **About this scenario.** Sometimes there are errors in communications: the server can disconnect a connection because it is not authorized or a message does not have the correct format. There are 2 events where errors are captured.

Delphi (VCL / FireMonkey)

```
procedure OnError(Connection: TsgcWSConnection; const Error: string);
begin
WriteLn('#error: ' + Error);
end;
```

C++ Builder

```
void OnError(TsgcWSConnection *Connection, const string Error)
{
WriteLn("#error: " + Error);
}
```

.NET (C#)

```
private void OnError(TsgcWSConnection Connection, string aError)
{
Console.WriteLine("#error: " + aError);
}
```

Common scenarios

The following scenarios are lifted verbatim from the online help. Each shows the configuration and method calls needed to drive the component through a specific real-world flow.

1 · TsgcWebSocketClient | Client Proxies

TsgcWebSocket client supports connections through proxies. To configure a proxy connection, just fill in the Proxy properties of the TsgcWebSocket client.

Delphi (VCL / FireMonkey)

```
Client := TsgcWebSocketClient.Create(nil);
Client.Proxy.Enabled := true;
Client.Proxy.Username := 'user';
Client.Proxy.Password := 'secret';
Client.Proxy.Host := '80.55.44.12';
Client.Proxy.Port := 8080;
Client.Active := True;
```

C++ Builder

```
Client = new TsgcWebSocketClient();
Client->Proxy->Enabled = true;
Client->Proxy->Username = "user";
Client->Proxy->Password = "secret";
Client->Proxy->Host = "80.55.44.12";
Client->Proxy->Port = 8080;
Client->Active = true;
```

.NET (C#)

```
Client = new TsgcWebSocketClient();
Client.Proxy.Enabled = true;
Client.Proxy.Username = "user";
Client.Proxy.Password = "secret";
Client.Proxy.Host = "80.55.44.12";
Client.Proxy.Port = 8080;
Client.Active = true;
```

2 · TsgcWebSocketClient | Client Register Protocol

By default, TsgcWebSocketClient does not use any SubProtocol. WebSocket sub-protocols are built on top of the WebSocket protocol and define a custom message protocol. Examples of WebSocket sub-protocols include MQTT, STOMP, etc.

Delphi (VCL / FireMonkey)

```
Client := TsgcWebSocketClient.Create(nil);
Client.Host := 'server host';
Client.Port := server.port;
Client.RegisterProtocol('myprotocol');
Client.Active := True;
```

C++ Builder

```
Client = new TsgcWebSocketClient();
Client→Host = "server host";
Client→Port = server.port;
Client→RegisterProtocol("myprotocol");
Client→Active = true;
```

.NET (C#)

```
Client = new TsgcWebSocketClient();
Client.Host = "server host";
Client.Port = server.port;
Client.RegisterProtocol("myprotocol");
Client.Active = true;
```

3 · TsgcWebSocketClient | Client Send a Text and Binary Message

WebSocket protocol only allows two types of messages: Text or Binary. However, you cannot send binary data along with text in the same message.

Delphi (VCL / FireMonkey)

```
sgcWSStreamWrite('00001', oStream);
TsgcWebSocketClient1.WriteData(oStream);
```

C++ Builder

```
sgcWSStreamWrite("00001", oStream);
TsgcWebSocketClient1->WriteData(oStream);
```

```
.NET (C#)
```

```
sgcWSBytesWrite("00001", oBytes);
TsgcWebSocketClient1.WriteData(oBytes);
```

4 · TsgcWebSocketClient | Receive Binary Messages

When the client receives a Binary Message, the OnBinary event is fired. Read the Data parameter to retrieve the binary message received.

```
Delphi (VCL / FireMonkey)
```

```
procedure OnBinary(Connection: TsgcWSConnection; const Data: TMemoryStream);
var
oBitmap: TBitmap;
begin
oBitmap := TBitmap.Create;
Try
oBitmap.LoadFromStream(Data);
Image1.Picture.Assign(oBitmap);
Log(
'#image uncompressed size: ' + IntToStr(Data.Size) +
'. Total received: ' + IntToStr(Connection.RecBytes));
Finally
FreeAndNil(oBitmap);
End;
end;
```

```
C++ Builder
```

```
void OnBinary(TsgcWSConnection *Connection, const TMemoryStream *Data)
{
oBitmap = new TBitmap();
oBitmap->LoadFromStream(Data);
Image1->Picture->Assign(oBitmap);
Log(
"#image uncompressed size: " + IntToStr(Data->Size) +
". Total received: " + IntToStr(Connection->RecBytes));
delete oBitmap;
}
```

.NET (C#)

```
private void OnBinary(TsgcWSConnection Connection, byte[] Bytes)
{
MemoryStream stream = new MemoryStream(Bytes);
pictureBox1.Image = new Bitmap(stream);
}
```

5 · TsgcWebSocketClient | Receive Text Messages

When the client receives a Text Message, the OnMessage event is fired. Read the Text parameter to retrieve the string of the message received.

Delphi (VCL / FireMonkey)

```
procedure OnMessage(Connection: TsgcWSConnection; const Text: string);
begin
ShowMessage('Message Received from Server: ' + Text);
end;
```

C++ Builder

```
void OnMessage(TsgcWSConnection *Connection, const string Text)
{
ShowMessage("Message Received from Server: " + Text);
}
```

.NET (C#)

```
void OnMessage(TsgcWSConnection Connection, string Text)
{
MessageBox.Show("Message Received from Server: " + Text);
}
```

6 · TsgcWebSocketClient | SChannel Get Connection Info

Once the client has connected to the secure server, you can request information about which TLS version is being used (TLS 1.2, TLS 1.3, etc.), the cipher used, strength, and more.

Delphi (VCL / FireMonkey)

```
uses
 </code><span>sgcIdSSL, sgcSSL_SChannel_Indy, sgcSSL_SChannel;
</span>
<code class="delphi">var
SSL: TsgcIdSSLIOHandlerSocketSChannel;
oClient := TsgcWebSocketClient.Create(nil);
oClient.URL := 'wss://www.esegece.com:2053';
oClient.TLSOptions.Version := tls1_2;
oClient.TLSOptions.IOHandler := iohSChannel;
oClient.OnSSLAfterCreateHandler := OnSSLAfterCreateHandlerEvent;
oClient.OnConnect := OnConnectEvent;
oClient.Active := True;
procedure OnSSLAfterCreateHandlerEvent(Sender: TObject; aType: TwsSSLHandler;
 aSSLHandler: TIdSSLIOHandlerSocketBase);
begin
if aSSLHandler.ClassType = TsgcIdSSLIOHandlerSocketSChannel then
SSL := TsgcIdSSLIOHandlerSocketSChannel(aSSLHandler);
end;
procedure OnConnectEvent(Connection: TsgcWSConnection);
var
oInfo: TsgcSChannelConnectionInfo;
begin
if Assigned(SSL) then
begin
oInfo := SSL.GetInfo;
if (oInfo.Protocol ≠ tls1_2) then
raise Exception.Create('Client cannot connect using TLS 1.2');
end;
end;
```

C++ Builder

```
oClient = new TsgcWebSocketClient();
oClient->URL = "wss://www.esegece.com:2053";
oClient->TLSOptions->Version = tls1_2;
oClient->TLSOptions->IOHandler = iohSChannel;
oClient->OnSSLAfterCreateHandler = OnSSLAfterCreateHandlerEvent;
oClient->OnConnect = OnConnectEvent;
oClient->Active = true;
void OnSSLAfterCreateHandlerEvent(TObject *Sender, TwsSSLHandler aType,
 TIdSSLIOHandlerSocketBase *aSSLHandler)
{
if (aSSLHandler->ClassType() == __classid(TsgcIdSSLIOHandlerSocketSChannel))
{
SSL = dynamic_cast<TsgcIdSSLIOHandlerSocketSChannel*>(aSSLHandler);
}
}
void OnConnectEvent(TsgcWSConnection *Connection)
{
if (SSL != NULL)
{
TsgcSChannelConnectionInfo oInfo = SSL->GetInfo();
if (oInfo->Protocol != tls1_2)
{
throw Exception("Client cannot connect using TLS 1.2");
}
}
}
```

.NET (C#)

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 — WebSocket Protocol — RFC 6455	datatracker.ietf.org/doc/html/rfc6455
Primary standard / spec — Per-message Deflate — RFC 7692	datatracker.ietf.org/doc/html/rfc7692
Online help — component page	www.esegece.com/help/sgcWebSockets/Components/TsgcWebSocketClient.htm
Delphi demo project (in the sgcWebSockets package)	`Demos\01.WebSocket_Quick_Start\02.WebSocket_Clients_APIs`
Component page	www.esegece.com/products/websockets/components/delphi/websocket-client/
Product page	www.esegece.com/products/websockets/

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