SGCWEBSOCKETS · TECHNICAL DOCUMENT

HTTP/2 Server

TsgcWebSocketHTTPServer with HTTP/2 — RFC 9113 server-side HTTP/2 with ALPN, HPACK and server push.

eSeGeCe

WWW.ESEGECE.COM

Overview

TsgcWebSocketHTTPServer implements Server WebSocket Component and can handle multiple threaded client connections as TsgcWebSocketServer, and allows you to serve HTML pages using a built-in HTTP Server, sharing the same port for WebSocket connections and HTTP requests.

At a glance

COMPONENT CLASS

`TsgcWebSocketHTTPServer`

STANDARDS / SPEC

HTTP/2 — RFC 9113

TRANSPORTS

TCP, TLS, HTTP, HTTPS

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	HTTP/2 — RFC 9113 · HPACK — RFC 7541
Component class	`TsgcWebSocketHTTPServer` (unit `sgcWebSocket_Server`)
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.

`APIKeyManager`	Optional API-key manager component used to validate incoming API keys before accepting a connection.
`Authentication`	Enables and configures user/password authentication for incoming WebSocket and HTTP connections.
`Active`	Starts or stops the HTTP/WebSocket server, opening the listening sockets on the configured bindings.
`Port`	TCP port on which the server accepts incoming HTTP and WebSocket connections.
`HTTP2Options`	Enables and tunes the HTTP/2 protocol handler used to serve HTTPS requests.
`SSLOptions`	Holds certificate paths, TLS version selection and OpenSSL tuning for the TLS listener.
`SecurityOptions`	Defines admission rules such as allowed origins for browser WebSocket handshakes.
`HeartBeat`	Sends periodic ping frames to keep idle client connections alive and detect dead peers.
`WatchDog`	Automatically restarts the server after an unexpected shutdown or listener failure.

`Options`	Bundles miscellaneous server behaviour flags: fragment handling, timeouts, HTTP test pages and UTF-8 validation.

Main methods

The principal public methods exposed by the component.

`Start()`	Starts the HTTP server from a secondary thread so the calling thread is not blocked while bindings are opened.
`Stop()`	Stops the HTTP server from a secondary thread so the calling thread is not blocked while connections are closed.
`ReStart()`	Stops and then restarts the server from a secondary thread, useful after changing bindings or ports at runtime.
`DisconnectAll()`	Disconnects every active client connection while keeping the server listening for new connections.
`WriteData()`	Sends a WebSocket message to a single client identified by its connection GUID.
`Ping()`	Sends a WebSocket ping frame to every connected WebSocket client.
`Broadcast()`	Sends the same WebSocket message to all connected clients, optionally filtered by channel, protocol, or connection GUID list.
`PushPromiseAddPreLoadLinks()`	Registers an HTTP/2 Server Push rule that preloads a set of related resources whenever a matching request path is served.
`PushPromiseRemovePreLoadLinks()`	Removes the HTTP/2 Server Push rule previously registered for the given request path.

Public events

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

`OnAfterForwardHTTP`	Fires after an HTTP request has been forwarded so the application can inspect the result or an error returned by the upstream server.
`OnAuthentication`	Fires when authentication is enabled so the application can check user and password and accept or reject the

	connection.
`OnBeforeCommand`	Fires before OnCommandGet or OnCommandOther so the request can be screened, authorized, or short-circuited with a 401 response.
`OnBeforeForwardHTTP`	Fires before an HTTP request is dispatched so it can be forwarded (reverse-proxied) to another HTTP server.
`OnBeforeHeartBeat`	Fires before each HeartBeat ping so the application can implement a custom keep-alive.
`OnBinary`	Fires every time a client sends a binary message and it is received by the server.
`OnCommandGet`	Fires when the HTTP server receives a GET, POST, or HEAD request so the application can build the response.
`OnCommandOther`	Fires when the HTTP server receives a method other than GET, POST or HEAD (PUT, DELETE, OPTIONS, PATCH...).
`OnConnect`	Fires every time a WebSocket connection is established with a client.
`OnCreateSession`	Fires when the HTTP server needs to create a new session so the application can supply a custom TIdHTTPSession instance.
`OnDisconnect`	Fires every time a WebSocket connection with a client is dropped.
`OnError`	Fires whenever a WebSocket protocol error occurs, such as a mal-formed handshake.
`OnException`	Fires whenever an unhandled exception is raised while processing a client connection.
`OnFragmented`	Fires when a fragment of a message is received (only when Options.FragmentedMessages is frgAll or frgOnlyFragmented).
`OnHTTP2BeforeAsyncRequest`	TsgcWebSocketHTTPServer › Events › OnHTTP2BeforeAsyncRequest
`OnHTTPUploadAfterSaveFile`	TsgcWebSocketHTTPServer › Events › OnHTTPUploadAfterSaveFile

`OnHTTPUploadBeforeCreatePostStream`	TsgcWebSocketHTTPServer › Events › OnHTTPUploadBeforeCreatePostStream
`OnHTTPUploadBeforeSaveFile`	TsgcWebSocketHTTPServer › Events › OnHTTPUploadBeforeSaveFile
`OnHTTPUploadReadInput`	Fires when the multipart/form-data decoder reads a non-file input field so its value can be captured.
`OnHandshake`	Fires after the handshake is evaluated on the server side and before the response is sent.
`OnInvalidSession`	Fires when an HTTP request presents an unknown or expired session ID so the application can decide how to react.
`OnLoadBalancerConnect`	property OnLoadBalancerConnect: TsgcWSConnectEvent; // TsgcWSConnectEvent = procedure(Connection: TsgcWSConnection) of object __property TsgcWSConnectEvent OnLoadBalancerConnect; // typedef void __fas...
`OnLoadBalancerDisconnect`	TsgcWebSocketHTTPServer › Events › OnLoadBalancerDisconnect
`OnLoadBalancerError`	Fires when an error occurs communicating with the Load Balancer Server.
`OnMessage`	Fires every time a client sends a text message and it is received by the server.
`OnSSLALPNSelect`	Fires during an ALPN-enabled handshake so the application can pick which protocol to negotiate.
`OnSSLAfterCreateHandler`	TsgcWebSocketHTTPServer › Events › OnSSLAfterCreateHandler
`OnSSLGetHandler`	Fires before the SSL handler is created so a custom server-side handler instance can be supplied.
`OnSSLVerifyPeer`	Fires when VerifyCertificate is enabled and the client presents a certificate to be accepted or rejected.
`OnSessionEnd`	Fires when an HTTP session is closed, either explicitly or after SessionTimeout expires.

`OnSessionStart`	Fires when an HTTP session is started and added to the SessionList.
`OnShutdown`	Fires after the server has stopped and no more connections are accepted.
`OnStartup`	Fires after the server has started and is ready to accept connections.
`OnTCPConnect`	Fires after a client connects at TCP level and before the WebSocket handshake, so the connection can be accepted or rejected.
`OnUnknownAuthentication`	TsgcWebSocketHTTPServer › Events › OnUnknownAuthentication
`OnUnknownProtocol`	Fires when the first message does not match a known protocol so the connection can be accepted or rejected.

Quick Start

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

> **About this scenario.** If server returns a header requesting Basic Authentication, set OnHTTP2Authorization the username and password.

Delphi (VCL / FireMonkey)

```
oClient := TsgcHTTP2Client.Create(nil);
oClient.OnHTTP2Authorization := OnHTTP2AuthorizationEvent;
...
procedure OnHTTP2AuthorizationEvent(Sender: TObject; const Connection: TsgcHTTP2ConnectionClient
begin
if AuthType = 'Basic' then
begin
UserName := 'user';
        Password := 'secret';
end;
end;
```

C++ Builder

```
TsgcHTTP2Client *oClient = new TsgcHTTP2Client();
oClient->OnHTTP2Authorization = OnHTTP2AuthorizationEvent;
...
void OnHTTP2AuthorizationEvent(TObject *Sender, const TsgcHTTP2ConnectionClient *Connection, con
{
if (AuthType == "Basic")
{
UserName = "user";
        Password = "secret";
}
}
```

.NET (C#)

```
TsgcHTTP2Client oClient = new TsgcHTTP2Client();
oClient.OnHTTP2Authorization += OnHTTP2AuthorizationEvent;
...
void OnHTTP2AuthorizationEvent(object Sender, TsgcHTTP2ConnectionClient Connection, string AuthT
{
if (AuthType == "Basic")
{
UserName = "user";
        Password = "secret";
}
}
```

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 · Asynchronous Mode

Get the following url: https://www.google.com and be notified when client receives the full response. After you call GETASYNC method, the process continues and OnHTTP2Response event is called when response is received.

Delphi (VCL / FireMonkey)

```
oClient := TsgcHTTP2Client.Create(nil);
oClient.OnHTTP2Response := OnHTTP2ResponseEvent;
oClient.GetAsync('https://www.gooogle.com');
procedure OnHTTP2ResponseEvent(Sender: TObject; const Connection: TsgcHTTP2ConnectionClient;
 const Request: TsgcHTTP2RequestProperty; const Response: TsgcHTTP2ResponseProperty);
begin
ShowMessage(Response.Headers.Text + #13#10 + Response.DataString);
end;
```

C++ Builder

```
TsgcHTTP2Client *oClient = new TsgcHTTP2Client();
oClient->OnHTTP2Response = OnHTTP2ResponseEvent;
oClient->GetAsync("https://www.gooogle.com");
void OnHTTP2ResponseEvent(TObject *Sender, const TsgcHTTP2ConnectionClient *Connection,
 const TsgcHTTP2RequestProperty *Request, const TsgcHTTP2ResponseProperty *Response)
{
ShowMessage(Response->Headers->Text + #13#10 + Response->DataString);
}
```

.NET (C#)

```
TsgcHTTP2Client oClient = new TsgcHTTP2Client();
oClient.OnHTTP2Response += OnHTTP2ResponseEvent;
oClient.GetAsync("https://www.goooogle.com");
void OnHTTP2ResponseEvent(object Sender, TsgcHTTP2ConnectionClient Connection,
  TsgcHTTP2RequestProperty Request, TsgcHTTP2ResponseProperty Response)
{
MessageBox.Show(Response.Headers.Text + #13#10 + Response.DataString);
}
```

2 · Configure JWT Client

Configure the JWT Client with the following values:

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

```
oHTTP := TsgcHTTP2Client.Create(nil);
oHTTP.TLSOptions.IOHandler := iohOpenSSL;
<br/>

oJWT := TsgcHTTP_JWT_Client.Create(nil);
oHTTP.Authentication.Token.JWT := oJWT;
oJWT.JWTOptions.Header.alg := jwtES256;
oJWT.JWTOptions.Header.kid := 'apple key id';
oJWT.JWTOptions.Payload.iss := 'issuer';
oJWT.JWTOptions.Payload.iat := StrToInt64(GetDateTimeUnix(Now, False));
oJWT.JWTOptions.Algorithms.ES.PrivateKey.LoadFromFile('AuthKey_**.p8');
oJWT.JWTOptions.RefreshTokenAfter := 60*40;
<br/>

oHTTP.Request.CustomHeaders.Clear;
oHTTP.Request.CustomHeaders.Add('apns-topic: com.example.application');
```

```
C++ Builder
```

```
TsgcHTTP2Client *oHTTP = new TsgcHTTP2Client(NULL);
oHTTP->TLSOptions->IOHandler = iohOpenSSL;
<br/>

TsgcHTTP_JWT_Client *oJWT = new TsgcHTTP_JWT_Client(NULL);
oHTTP->Authentication->Token->JWT = oJWT;
oJWT->JWTOptions->Header->alg = jwtES256;
oJWT->JWTOptions->Header->kid = "apple key id";
oJWT->JWTOptions->Payload->iss = "issuer";
oJWT->JWTOptions->Payload->iat = StrToInt64(GetDateTimeUnix(Now, False));
oJWT->JWTOptions->Algorithms->ES->PrivateKey->LoadFromFile("AuthKey_**.p8");
oJWT->JWTOptions->RefreshTokenAfter = 60*40;
<br/>

oHTTP->Request->CustomHeaders->Clear();
oHTTP->Request->CustomHeaders->Add("apns-topic: com.example.application");
```

.NET (C#)

```
TsgcHTTP2Client oHTTP = new TsgcHTTP2Client();
oHTTP.TLSOptions.IOHandler = TwsTLSIOHandler.iohOpenSSL;
<br/>

TsgcHTTP_JWT_Client oJWT = new TsgcHTTP_JWT_Client();
oHTTP.Authentication.Token.JWT = oJWT;
oJWT.JWTOptions.Header.alg = TsgcHTTP_JWT_Algorithm.jwtES256;
oJWT.JWTOptions.Header.kid = "apple key id";
oJWT.JWTOptions.Payload.iss = "issuer";
oJWT.JWTOptions.Payload.iat = unix time;
oJWT.JWTOptions.Algorithms.ES.PrivateKey.LoadFromFile("AuthKey_**.p8");
oJWT.JWTOptions.RefreshTokenAfter = 60*40;
<br/>

oHTTP.Request.CustomHeaders = "apns-topic: com.example.application";
```

3 · HeartBeat

HeartBeat property allows you to send a Ping every X seconds to maintain connection alive. Some servers, close TCP connections if there is no data exchanged between peers. HeartBeat solves this problem, sending a ping every a specific interval. Usually this is enough to maintain a connection active.

Delphi (VCL / FireMonkey)

```
oClient := TsgcHTTP2Client.Create(nil);
oClient.HeartBeat.Interval := 30;
oClient.HeartBeat.Enabled := true;
oClient.Active := true;
```

C++ Builder

```
oClient = new TsgcHTTP2Client();
oClient->HeartBeat->Interval = 30;
oClient->HeartBeat->Enabled = true;
oClient->Active = true;
```

.NET (C#)

```
oClient = new TsgcHTTP2Client();
oClient.HeartBeat.Interval = 30;
oClient.HeartBeat.Enabled = true;
oClient.Active = true;
```

4 · Large File Downloads

When downloading large files (hundreds of MB or more), set HTTP2Options.ReadTimeout to 0 (no timeout) to ensure the transfer completes without being interrupted by the default 60-second timeout:

Delphi (VCL / FireMonkey)

```
oClient := TsgcHTTP2Client.Create(nil);
oClient.HTTP2Options.ReadTimeout := 0; // no timeout for large files
oClient.Get('https://server/largefile', oStream);
```

C++ Builder

```
TsgcHTTP2Client *oClient = new TsgcHTTP2Client();
oClient->HTTP2Options->ReadTimeout = 0; // no timeout for large files
oClient->Get("https://server/largefile", oStream);
```

.NET (C#)

```
TsgcHTTP2Client oClient = new TsgcHTTP2Client();
oClient.HTTP2Options.ReadTimeout = 0; // no timeout for large files
oClient.Get("https://server/largefile", oStream);
```

5 · OpenSSL

If you use OpenSSL, you must deploy the OpenSSL libraries with your application. Before setting the certificate with the TsgcHTTP2Client, this certificate must first be converted to PEM format because OpenSSL doesn't allow importing P12 certificates directly.

Delphi (VCL / FireMonkey)

```
oHTTP := TsgcHTTP2Client.Create(nil);
oHTTP.TLSOptions.IOHandler := iohOpenSSL;
oHTTP.TLSOptions.CertFile := 'certificate_file.pem';
oHTTP.TLSOptions.KeyFile := 'private_key.pem';
oHTTP.TLSOptions.Password := 'certificate password';
oHTTP.TLSOptions.Version := tls1_2;
```

C++ Builder

```
TsgcHTTP2Client *oHTTP = new TsgcHTTP2Client(NULL);
oHTTP->TLSOptions->IOHandler = iohOpenSSL;
oHTTP->TLSOptions->CertFile = "certificate_file.pem";
oHTTP->TLSOptions->KeyFile = "private_key.pem";
oHTTP->TLSOptions->Password = "certificate password";
oHTTP->TLSOptions->Version = tls1_2;
```

.NET (C#)

```
TsgcHTTP2Client oHTTP = new TsgcHTTP2Client();
oHTTP.TLSOptions.IOHandler = TwsTLSIOHandler.iohOpenSSL;
oHTTP.TLSOptions.CertFile = "certificate_file.pem";
oHTTP.TLSOptions.KeyFile = "private_key.pem";
oHTTP.TLSOptions.Password = "certificate password";
oHTTP.TLSOptions.Version = TwsTLSVersions.tls1_2;
```

6 · Requests | HTTP/2 Server Push

Server Push is the ability of the server to send multiple responses for a single client request. That is, in addition to the response to the original request, the server can push additional resources to the client, without the client having to request each one explicitly.

Delphi (VCL / FireMonkey)

```
oClient := TsgcHTTP2Client.Create(nil);
oClient.OnHTTP2PushPromise := OnHTTP2PushPromiseEvent;
oClient.Get('https://http2.golang.org/serverpush');
...
procedure OnHTTP2PushPromiseEvent(Sender: TObject; const Connection: TsgcHTTP2ConnectionClient;
 const PushPromise: TsgcHTTP2_Frame_PushPromise; var Cancel: Boolean);
begin
if PushPromise.URL = '/serverpush/static/godocs.js' then
Cancel := True
else
Cancel := False;
end;
```

C++ Builder

```
TsgcHTTP2Client *oClient = new TsgcHTTP2Client();
oClient->OnHTTP2PushPromise = OnHTTP2PushPromiseEvent;
oClient->Get("https://http2.golang.org/serverpush");
...
void OnHTTP2PushPromiseEvent(TObject *Sender, const TsgcHTTP2ConnectionClient *Connection,
 const TsgcHTTP2_Frame_PushPromise *PushPromise, bool &Cancel)
{
if (PushPromise->URL == "/serverpush/static/godocs.js")
{
Cancel = true;
}
else
{
Cancel = false;
}
}
```

.NET (C#)

```
TsgcHTTP2Client oClient = new TsgcHTTP2Client();
oClient.OnHTTP2PushPromise += OnHTTP2PushPromiseEvent;
oClient.Get("https://http2.golang.org/serverpush");
...
void OnHTTP2PushPromiseEvent(object Sender, TsgcHTTP2ConnectionClient Connection,
  TsgcHTTP2_Frame_PushPromise PushPromise, ref bool Cancel)
{
if (PushPromise.URL == "/serverpush/static/godocs.js")
{
Cancel = true;
}
else
{
Cancel = false;
}
}
```

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 — HTTP/2 — RFC 9113	datatracker.ietf.org/doc/html/rfc9113
Primary standard / spec — HPACK — RFC 7541	datatracker.ietf.org/doc/html/rfc7541
Online help — component page	www.esegece.com/help/sgcWebSockets/Components/TsgcWebSocketHTTPServer.htm
Delphi demo project (in the sgcWebSockets package)	`Demos\20.HTTP_Protocol\01.HTTP2_Server_And_Client`
Component page	www.esegece.com/products/websockets/http/http2-server/
Product page	www.esegece.com/products/websockets/

> **Document scope.** This document covers the publicly-documented surface of the HTTP/2 Server component shipped with sgcWebSockets. For full property, method and event reference consult the online help linked above.