SGCWEBSOCKETS · TECHNICAL DOCUMENT

HTTP/2 Client

TsgcHTTP2Client — RFC 9113 HTTP/2 client over TLS with ALPN, header compression (HPACK), multiplexing and stream prioritisation.

eSeGeCe

WWW.ESEGECE.COM

Overview

HTTP/2 is the binary successor to HTTP/1.1: a single multiplexed TLS connection carries any number of concurrent request / response streams, headers are compressed with HPACK, the server can push resources before the client asks for them, and stream priorities let the browser tell the server which response to deliver first. It is the wire protocol behind every modern browser, every public REST API at scale, and gRPC.

The sgcWebSockets `TsgcHTTP2Client` component is a native Delphi / C++ Builder / .NET HTTP/2 client implementation — no external libraries, no curl, no DLLs. It speaks RFC 9113 binary framing, HPACK header compression, ALPN protocol negotiation over TLS 1.2/1.3, and the full Get / Post / Put / Delete / Patch HTTP verb set, with both synchronous and asynchronous variants. Server push, stream priority and flow-control are exposed through events so your application can react in real time.

At a glance

COMPONENT CLASS `TsgcHTTP2Client`	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

- **RFC 9113 binary framing** — full HTTP/2 wire protocol, no external dependencies.
- **Multiplexed streams** — many concurrent requests over a single TLS connection without head-of-line blocking.
- **HPACK header compression** — static and dynamic table per RFC 7541.

- **ALPN over TLS 1.2 / 1.3** — protocol negotiation per RFC 7301, with OpenSSL and SChannel transports.
- **Synchronous and asynchronous requests** — `Get`, `Post`, `Put`, `Delete`, `Patch` plus their `*Async` counterparts; async returns immediately and fires `OnHTTP2Response` when the response arrives.
- **Server push (PUSH_PROMISE)** — receive pre-emptively pushed resources via `OnHTTP2PushPromise`; accept or cancel each push at the application level.
- **Streaming response fragments** — large bodies delivered chunk-by-chunk through `OnHTTP2ResponseFragment`; useful for downloads and SSE-style endpoints.
- **Authentication helpers** — Basic, Bearer (OAuth2 / JWT), Digest and custom-header schemes via `Authentication`.
- **Stream prioritisation, flow control, settings** — advertise window sizes, max concurrent streams, header table sizes; observe `OnHTTP2GoAway` and `OnHTTP2RSTStream` to react to peer-initiated tear-downs.
- **Connection events** — `OnHTTP2Connect`, `OnHTTP2Disconnect`, `OnHTTP2Exception`, `OnHTTP2PendingRequests` let you drive reconnect / retry logic.

Technical specification

Standards & specs	HTTP/2 — RFC 9113 · HPACK — RFC 7541 · TLS ALPN — RFC 7301
Component class	`TsgcHTTP2Client` (unit `sgcHTTP2_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.

`Authentication`	Configures the credentials used to authenticate HTTP/2 requests, including OAuth2 and JWT tokens.
`TLSOptions`	Configures certificates, TLS version, ALPN, IOHandler and other secure-connection details used for HTTP/2 over TLS.
`Active`	Opens or closes the HTTP/2 connection to the remote server.
`Host`	IP address or DNS name of the HTTP/2 server the client will connect to.
`Port`	TCP port used to connect to the HTTP/2 server.
`TLS`	Enables a secure TLS connection, which is normally required by HTTP/2 servers.
`Proxy`	Routes the HTTP/2 connection through an HTTP CONNECT tunnel or SOCKS proxy server.
`HeartBeat`	Sends periodic HTTP/2 PING frames to keep the connection alive.
`WatchDog`	Automatically reconnects to the HTTP/2 server after an unexpected disconnection.
`Throttle`	Limits the number of bits per second sent or received by the HTTP/2 socket.

Main methods

The principal public methods exposed by the component.

`ConnectAsync()`	Opens the HTTP/2 connection and issues a non-blocking GET; the reply is delivered on OnHTTP2Response.
`Get()`	Sends a synchronous HTTP/2 GET request and returns the response body.
`PostAsync()`	Sends a non-blocking POST; the reply arrives on OnHTTP2Response.
`PutAsync()`	Sends a non-blocking PUT; the reply arrives on OnHTTP2Response.
`DeleteAsync()`	Sends a non-blocking DELETE; the reply arrives on OnHTTP2Response.
`PatchAsync()`	Sends a non-blocking PATCH; the reply arrives on OnHTTP2Response.
`Connect()`	Opens the HTTP/2 connection and issues a synchronous GET to the supplied URL.
`Disconnect()`	Closes the underlying TCP/TLS socket immediately without sending a GOAWAY frame.
`Close()`	Performs a graceful shutdown by sending a GOAWAY frame with an error code and optional debug text.
`Ping()`	Sends an HTTP/2 PING frame to probe liveness and measure round-trip time.

Public events

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

`OnHTTP2Authorization`	Fires when the server requires authentication so the application can supply credentials or a bearer token.
`OnHTTP2BeforeRequest`	Fires just before request headers are sent so the application can add or modify them.
`OnHTTP2Connect`	Fires just after the client connects successfully to the HTTP/2 server.
`OnHTTP2Disconnect`	property OnHTTP2Disconnect: TsgcHTTP2ClientDisconnectEvent; // TsgcHTTP2ClientDisconnectEvent = procedure(Sender: TObject; const Connection: TsgcHTTP2ConnectionClient) of object __property TsgcHTTP2Cl...

`OnHTTP2Exception`	Fires when an exception is raised on the HTTP/2 connection so the application can handle it.
`OnHTTP2GoAway`	Fires when the server sends a GoAway frame signalling the connection is being shut down.
`OnHTTP2PendingRequests`	Fires after a disconnection when there are pending requests so the application can reconnect or clear the queue.
`OnHTTP2PushPromise`	Fires when the server pushes a resource so the client can accept or cancel it.
`OnHTTP2RSTStream`	property OnHTTP2RSTStream: TsgcHTTP2ClientRSTStreamEvent; // TsgcHTTP2ClientRSTStreamEvent = procedure(Sender: TObject; const Connection: TsgcHTTP2ConnectionClient; const RSTStream: TsgcHTTP2RSTStream...
`OnHTTP2Response`	Fires when the client receives the full response (status, headers and body) from the server.
`OnHTTP2ResponseFragment`	Fires for each streamed response fragment when FragmentedData delivers data as it arrives.
`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.

Quick Start

Drop the component on a form, configure the properties below and activate it. The snippet that follows shows the typical **TsgcHTTP2Client** configuration sourced from the online help.

> **About this scenario.** TsgcHTTP2Client implements Client HTTP/2 Component and can connect to HTTP/2 servers.

Delphi (VCL / FireMonkey)

```
oClient := TsgcHTTP2Client.Create(nil);
    oClient.OnHTTP2Response := OnHTTP2ResponseEvent;
    oClient.Get('https://www.google.com');

    procedure OnHTTP2ResponseEvent(Sender: TObject; const
    Connection: TsgcHTTP2ConnectionClient; const Request:
    TsgcHTTP2RequestProperty; const Response: TsgcHTTP2ResponseProperty);
    begin
    ShowMessage(Response.DataString);
    end;
```

C++ Builder

```
TsgcHTTP2Client *oClient = new TsgcHTTP2Client();
    oClient->OnHTTP2Response = OnHTTP2ResponseEvent;
    oClient->Get("https://www.google.com");

    void OnHTTP2ResponseEvent(TObject *Sender, const
    TsgcHTTP2ConnectionClient *Connection, const TsgcHTTP2RequestProperty *Request,
        const TsgcHTTP2ResponseProperty *Response)
    {
    ShowMessage(Response->DataString);
    }
```

.NET (C#)

```
TsgcHTTP2Client oClient = new TsgcHTTP2Client();
    oClient.OnHTTP2Response += OnHTTP2ResponseEvent;
    oClient.Get("https://www.google.com");

    void OnHTTP2ResponseEvent(object Sender,
        TsgcHTTP2ConnectionClient Connection, TsgcHTTP2RequestProperty Request,
        TsgcHTTP2ResponseProperty Response)
    {
    MessageBox.Show(Response.DataString);
    }
```

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 · TsgcHTTP2Client | HTTP/2 Headers

TsgcHTTP2Client allows customizing Headers sent to server when client connects

Delphi (VCL / FireMonkey)

```
procedure OnHTTP2BeforeRequest(Sender: TObject; const Connection: TsgcHTTP2ConnectionClient;
 var Headers: TStringList);
begin
Headers.Add('Client: sgcWebSockets');
end;
```

C++ Builder

```
void OnHTTP2BeforeRequest(TObject *Sender, const TsgcHTTP2ConnectionClient *Connection,
 ref TStringList *Headers)
{
Headers->Add("Client: sgcWebSockets");
}
```

.NET (C#)

```
void OnHTTP2BeforeRequest(object Sender, TsgcHTTP2ConnectionClient Connection,
  ref string Headers)
{
Headers = Headers + Environment.NewLine + "Client: sgcWebSockets";
}
```

2 · TsgcHTTP2Client | HTTP/2 Partial Responses

Usually when you send an HTTP Request, server sends a response with the file requested, sometimes, instead of sending a single response, the server can send multiple responses like a stream, in these cases you can use OnHTTP2ResponseFragment event to capture these responses and show to user.

Delphi (VCL / FireMonkey)

```
oClient := TsgcHTTP2Client.Create(nil);
oClient.OnHTTP2ResponseFragment := OnHTTP2ResponseFragmentEvent;
oClient.Get('https://http2.golang.org/clockstream');
...
procedure OnHTTP2ResponseFragmentEvent(Sender: TObject; const Connection: TsgcHTTP2ConnectionCli
 const Request: TsgcHTTP2RequestProperty; const Fragment: TsgcHTTP2ResponseFragmentProperty);
begin
ShowMessage(Fragment.DataString);
end;
```

C++ Builder

```
TsgcHTTP2Client *oClient = new TsgcHTTP2Client();
oClient->OnHTTP2ResponseFragment = OnHTTP2ResponseFragmentEvent;
oClient->Get("https://http2.golang.org/clockstream");
...
void OnHTTP2ResponseFragmentEvent(TObject *Sender, const TsgcHTTP2ConnectionClient *Connection,
 const TsgcHTTP2RequestProperty *Request, const TsgcHTTP2ResponseFragmentProperty *Fragment)
{
ShowMessage(Fragment->DataString);
}
```

.NET (C#)

```
TsgcHTTP2Client oClient = new TsgcHTTP2Client();
oClient.OnHTTP2ResponseFragment += OnHTTP2ResponseFragmentEvent;
oClient.Get("https://http2.golang.org/clockstream");
...
void OnHTTP2ResponseFragmentEvent(object Sender, TsgcHTTP2ConnectionClient Connection,
  TsgcHTTP2RequestProperty Request, TsgcHTTP2ResponseFragmentProperty Fragment)
{
MessageBox.Show(Fragment.DataString);
}
```

3 · Basic Authentication

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";
}
}
```

4 · 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.goooogle.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.goooogle.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);
}
```

5 · 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;
```

6 · 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);
```

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
Primary standard / spec — TLS ALPN — RFC 7301	datatracker.ietf.org/doc/html/rfc7301
Online help — component page	www.esegece.com/help/sgcWebSockets/Components/HTTP/HTTP2/Client/TsgcHTTP2Client.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-client/
Product page	www.esegece.com/products/websockets/

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