SGCWEBSOCKETS · TECHNICAL DOCUMENT

WebSocket Server (Delphi)

TsgcWebSocketServer / TsgcWebSocketHTTPServer — Delphi WebSocket and HTTP server, RFC 6455 compliant, with TLS, virtual hosts and protocol pipelines.

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

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	`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 **TsgcWebSocketHTTPServer** configuration sourced from the online help.

> **About this scenario.** To improve the performance of the HTTP/2 protocol, the requests are dispatched by default in a pool of threads (by default 32) every time a new HTTP/2 request is received by the server. This avoids waits when a single connection sends many concurrent requests that would require sequential processing (in the context of the connection thread) in the absence of this pool of threads.

Delphi (VCL / FireMonkey)

```
procedure OnHTTP2BeforeAsyncRequest(Sender: TObject; Connection: TsgcWSConnection; const ARequest
begin
  if ARequestInfo.Document = '/fast-request' then
    ASync := False;
end;
```

C++ Builder

```
void __fastcall TForm1::OnHTTP2BeforeAsyncRequest(TObject *Sender, TsgcWSConnection *Connection,
{
    if (ARequestInfo.Document == "/fast-request")
        Async = false;
}
```

.NET (C#)

```
void OnHTTP2BeforeAsyncRequest(object Sender, TsgcWSConnection Connection, TIdHTTPRequestInfo AR
{
  if (ARequestInfo.Document == "/fast-request")
    Async = false;
}
```

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 · TsgcWebSocketHTTPServer | 404 Error without Response Body

By default, the Indy library adds a content body to HTTP responses if there is no ContentText or ContentStream assigned. If you want to return an empty response body (for a 404 error or similar), you can use the following approach.

Delphi (VCL / FireMonkey)

```
procedure OnCommandGet(AContext: TIdContext; ARequestInfo: TIdHTTPRequestInfo;
 AResponseInfo: TIdHTTPResponseInfo);
begin
AResponseInfo.ContentStream := TStringStream.Create('');
AResponseInfo.ContentType := 'text/html';
AResponseInfo.ResponseNo := 404;
end;
```

C++ Builder

```
private void OnCommandGet(TIdContext *AContext, TIdHTTPRequestInfo *ARequestInfo,
  TIdHTTPResponseInfo *AResponseInfo)
{
AResponseInfo->ContentStream = new TStringStream("");
AResponseInfo->ContentType = "text/html";
AResponseInfo->ResponseNo = 404;
}
```

.NET (C#)

```

```

2 · TsgcWebSocketHTTPServer | Stream Video

If you want to stream a video file using the server, you can use the function IndyStreamFileVideo to stream the file using chunked transfer encoding.

Delphi (VCL / FireMonkey)

```
procedure OnServerCommandGet(AContext: TIdContext; ARequestInfo: TIdHTTPRequestInfo; AResponseIn
begin
  if ARequestInfo.Document = '/video.mp4' then
    IndyStreamFileVideo(Acontext, AResponseInfo, 'C:\videos\video.mp4')
  else
    AResponseInfo.ResponseNo := 404;
end;
```

C++ Builder

```
void OnServerCommandGet(TIdContext *AContext, TIdHTTPRequestInfo *ARequestInfo, TIdHTTPResponseI
{
  if (ARequestInfo->Document == "/video.mp4") {
    IndyStreamFileVideo(Acontext, AResponseInfo, "C:\\videos\\video.mp4");
  } else {
    AResponseInfo->ResponseNo = 404;
  }
}
```

.NET (C#)

```
void OnServerCommandGet(TIdContext AContext, TIdHTTPRequestInfo ARequestInfo, TIdHTTPResponseInf
{
  if (ARequestInfo.Document == "/video.mp4")
  {
    IndyStreamFileVideo(AContext, AResponseInfo, "C:\\videos\\video.mp4");
  }
  else
  {
    AResponseInfo.ResponseNo = 404;
  }
}
```

3 · Certificate from Windows Store

If the certificate is already installed in the Windows Certificate Store, provide the certificate thumbprint and indicate where it is located.

Delphi (VCL / FireMonkey)

```
oServer := TsgcWebSocketHTTPServer.Create(nil);
oServer.SSL := True;
oServer.SSLOptions.IOHandler := iohSChannel;
oServer.SSLOptions.Version := tls1_2;
oServer.SSLOptions.Port := 443;
oServer.Port := 443;
oServer.SSLOptions.SChannel_Options.CertHash := 'C12A8FC8AE668F866B48F23E753C93D357E9BE10';
oServer.SSLOptions.SChannel_Options.CertStoreName := scsnMY;
oServer.SSLOptions.SChannel_Options.CertStorePath := scspStoreLocalMachine;
oServer.Active := True;
```

C++ Builder

```
oServer = new TsgcWebSocketHTTPServer();
oServer->SSL = true;
oServer->SSLOptions->IOHandler = iohSChannel;
oServer->SSLOptions->Version = tls1_2;
oServer->SSLOptions->Port = 443;
oServer->Port = 443;
oServer->SSLOptions->SChannel_Options->CertHash = "C12A8FC8AE668F866B48F23E753C93D357E9BE10";
oServer->SSLOptions->SChannel_Options->CertStoreName = scsnMY;
oServer->SSLOptions->SChannel_Options->CertStorePath = scspStoreLocalMachine;
oServer->Active = true;
```

.NET (C#)

```
TsgcWebSocketHTTPServer oServer = new TsgcWebSocketHTTPServer();
oServer.SSL = true;
oServer.SSLOptions.IOHandler = TsgcSSLIOHandler.iohSChannel;
oServer.SSLOptions.Version = TsgcTLSVersion.tls1_2;
oServer.SSLOptions.Port = 443;
oServer.Port = 443;
oServer.SSLOptions.SChannel_Options.CertHash = "C12A8FC8AE668F866B48F23E753C93D357E9BE10";
oServer.SSLOptions.SChannel_Options.CertStoreName = TsgcSChannelCertStoreName.scsnMY;
oServer.SSLOptions.SChannel_Options.CertStorePath = TsgcSChannelCertStorePath.scspStoreLocalMach
oServer.Active = true;
```

4 · Configure Server Push

Following the screenshots above, you can configure your server so that every time there is a new request for the /index.html file, the server will send index.html and styles.css.

Delphi (VCL / FireMonkey)

```
server := TsgcWebSocketHTTPServer.Create(nil);
oLinks := TStringList.Create;
Try
  oLinks.Add('/styles.css');
  server.PushPromiseAddPreLoadLinks('/index.html', oLinks);
Finally
  oLinks.Free;
End;

      procedure OnCommandGet(AContext: TIdContext; ARequestInfo: TIdHTTPRequestInfo; AResponseIn
TIdHTTPResponseInfo);
begin
  if ARequestInfo.Document = '/index.html' then
  begin
    AResponseInfo.ContentText := '';
    AResponseInfo.ContentType := 'text/html';
    AResponseInfo.ResponseNo := 200;
  end
  else if ARequestInfo.Document = '/styles.css' then
  begin
    AResponseInfo.ContentText := '';
    AResponseInfo.ContentType := 'text/css';
    AResponseInfo.ResponseNo := 200;
  end;
end;
```

C++ Builder

```
TsgcWebSocketHTTPServer *server = new TsgcWebSocketHTTPServer(this);
TStringList *oLinks = new TStringList();
try
{
  oLinks->Add("/styles.css");
  server->PushPromiseAddPreLoadLinks("/index.html", oLinks);
}
__finally
{
  oLinks->Free();
}

void OnCommandGet(TIdContext *AContext, TIdHTTPRequestInfo *ARequestInfo, TIdHTTPResponseInfo *A
{
  if (ARequestInfo->Document == "/index.html")
  {
    AResponseInfo->ContentText = "";
    AResponseInfo->ContentType = "text/html";
    AResponseInfo->ResponseNo = 200;
  }
  else if (ARequestInfo->Document == "/styles.css")
  {
    AResponseInfo->ContentText = "";
    AResponseInfo->ContentType = "text/css";
    AResponseInfo->ResponseNo = 200;
  }
}
```

.NET (C#)

```
TsgcWebSocketHTTPServer server = new TsgcWebSocketHTTPServer(this);
server.PushPromiseAddPreLoadLinks("/index.html", "/styles.css");
void OnCommandGet(TsgcWSConnection Connection, TsgcWSHTTPRequestInfo RequestInfo, ref TsgcWSHTTP
{
  if (RequestInfo.Document == "/index.html")
  {
    ResponseInfo.ContentText = "";
    ResponseInfo.ContentType = "text/html";
    ResponseInfo.ResponseNo = 200;
  }
  else if (RequestInfo.Document == "/styles.css")
  {
    ResponseInfo.ContentText = "";
    ResponseInfo.ContentType = "text/css";
    ResponseInfo.ResponseNo = 200;
  }
}
```

5 · HTTP Server Sessions — Configuration

There are some properties in TsgcWebSocketHTTPServer that enable/disable sessions in the server component. The most important are:

Delphi (VCL / FireMonkey)

```
TsgcWebSocketHTTPServer1.SessionState := True;
TsgcWebSocketHTTPServer1.SessionTimeout := 600000;
AutoStartSession := False;
```

C++ Builder

```
TsgcWebSocketHTTPServer1->SessionState = true;
TsgcWebSocketHTTPServer1->SessionTimeout = 600000;
AutoStartSession = false;
```

.NET (C#)

```
TsgcWebSocketHTTPServer1.SessionState = true;
TsgcWebSocketHTTPServer1.SessionTimeout = 600000;
AutoStartSession = false;
```

6 · OnBeforeCommand

Use this event to customize the HTTP response. For example, if you want some endpoints to use an authorization scheme while others can be accessed without authorization, use the options parameter to allow or disable it. Below is an example where Authorization Basic is enabled, but when a user requests the endpoint /public, authorization is not required.

Delphi (VCL / FireMonkey)

```
procedure OnBeforeCommand(const aConnection: TsgcWSConnection; ARequestInfo: TIdHTTPRequestInfo;
begin
  if aRequestInfo.Document = '/public' then
    aOptions := [hcoAuthorizedBasic];
end;
```

C++ Builder

```
void __fastcall TForm1::OnBeforeCommand(TsgcWSConnection* aConnection, TIdHTTPRequestInfo* ARequ
{
  if (ARequestInfo->Document == "/public")
    aOptions << hcoAuthorizedBasic;
}
```

.NET (C#)

```
public void OnBeforeCommand(TsgcWSConnection aConnection, TIdHTTPRequestInfo ARequestInfo, TIdHT
{
    if (ARequestInfo.Document == "/public")
        aOptions = TsgcHTTPCommandOptions.hcoAuthorizedBasic;
}
```

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/TsgcWebSocketHTTPServer.htm
Delphi demo project (in the sgcWebSockets package)	`Demos\01.WebSocket_Quick_Start\01.Server_and_Client_Chat`
Component page	www.esegece.com/products/websockets/components/delphi/websocket-server/
Product page	www.esegece.com/products/websockets/

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