SGCWEBSOCKETS · TECHNICAL DOCUMENT

Load Balancer

TsgcWebSocketLoadBalancerServer — pluggable load balancer for sgcWebSockets back-ends with sticky sessions and health checks.

eSeGeCe

WWW.ESEGECE.COM

Overview

The component TsgcWebSocketLoadBalancerServer allows you to load-balance WebSocket and HTTP protocols. For the WebSocket protocol, it distributes messages across a group of servers and distributes client connections using a random sequence or fewest-connections algorithm.

At a glance

COMPONENT CLASS `TsgcWebSocketLoadBalancerServer`	STANDARDS / SPEC —
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

Component class	`TsgcWebSocketLoadBalancerServer` (unit `sgcWebSocket_LoadBalancer`)
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.

`Active`	Starts or stops the load balancer, opening the listening sockets that accept downstream clients and backend TsgcWebSocketServer registrations.
`Port`	TCP port on which the load balancer accepts incoming WebSocket/HTTP clients and backend server registrations.
`HTTP2Options`	Enables and tunes HTTP/2 on the load balancer's TLS listener used to serve HTTPS requests from downstream clients.
`SSLOptions`	Holds certificate paths, TLS version selection and OpenSSL tuning for the load balancer's TLS listener.
`SecurityOptions`	Defines admission rules such as allowed origins for WebSocket handshakes that reach the load balancer.
`Options`	Miscellaneous behaviour flags for the load balancer: fragment handling, timeouts, HTTP test pages and UTF-8 validation.
`ThreadPoolOptions`	Configures the size and upper bound of the reusable thread pool used when ThreadPool is enabled.
`Throttle`	Caps the bandwidth (bits per second) that the load balancer reads from or writes to each connection.
`Bindings`	Collection of IP/Port pairs the load balancer listens on for downstream clients and backend registrations.

`MaxConnections`	Maximum number of concurrent TCP connections (downstream clients plus registered backends) accepted by the load balancer.

Main methods

The principal public methods exposed by the component.

`Start()`	Starts the load balancer from a secondary thread so the calling thread is not blocked while bindings are opened.
`Stop()`	Stops the load balancer from a secondary thread so the calling thread is not blocked while connections are closed.
`ReStart()`	Stops and then restarts the load balancer from a secondary thread, useful after changing bindings or ports at runtime.
`DisconnectAll()`	Disconnects every active client connection and every registered backup server while keeping the load balancer listening for new connections.
`WriteData()`	Sends a WebSocket message to a single client identified by its connection GUID, routing the frame through the backup server that owns the session.
`Ping()`	Sends a WebSocket ping frame to every client connected through the load balancer.
`Broadcast()`	Fans a WebSocket message out across every backup server in the cluster, 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 by the load balancer.
`PushPromiseRemovePreLoadLinks()`	Removes the HTTP/2 Server Push rule previously registered on the load balancer for the given request path.

Public events

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

`OnBeforeSendServerBinding`	TsgcWebSocketLoadBalancerServer › Events › OnBeforeSendServerBinding

`OnBinary`	Fires when the load balancer itself receives a binary WebSocket frame on one of its downstream sessions.
`OnClientBinary`	TsgcWebSocketLoadBalancerServer › Events › OnClientBinary
`OnClientConnect`	TsgcWebSocketLoadBalancerServer › Events › OnClientConnect
`OnClientDisconnect`	TsgcWebSocketLoadBalancerServer › Events › OnClientDisconnect
`OnClientFragmented`	TsgcWebSocketLoadBalancerServer › Events › OnClientFragmented
`OnClientMessage`	TsgcWebSocketLoadBalancerServer › Events › OnClientMessage
`OnConnect`	Fires when a WebSocket connection (client or backend server) is established with the load balancer.
`OnDisconnect`	Fires when any WebSocket connection accepted by the load balancer is closed.
`OnError`	Fires when the load balancer detects an error on one of its accepted connections.
`OnException`	Fires when an unhandled Delphi exception is caught by the load balancer while processing a connection.
`OnFragmented`	Fires when the load balancer receives a fragmented WebSocket frame on one of its own sessions.
`OnHandshake`	Fires after the load balancer validates an incoming WebSocket handshake and before the HTTP response is returned.
`OnLoadBalancerHTTPRequest`	TsgcWebSocketLoadBalancerServer › Events › OnLoadBalancerHTTPRequest
`OnLoadBalancerHTTPResponse`	TsgcWebSocketLoadBalancerServer › Events › OnLoadBalancerHTTPResponse
`OnMessage`	Fires when the load balancer receives a text WebSocket frame on one of its own sessions.
`OnRawMessage`	Fires when any WebSocket text frame arrives, before higher-level protocols or the forwarder process it.
`OnSSLAfterCreateHandler`	TsgcWebSocketLoadBalancerServer › Events › OnSSLAfterCreateHandler

`OnSSLGetHandler`	TsgcWebSocketLoadBalancerServer › Events › OnSSLGetHandler
`OnServerConnect`	TsgcWebSocketLoadBalancerServer › Events › OnServerConnect
`OnServerDisconnect`	TsgcWebSocketLoadBalancerServer › Events › OnServerDisconnect
`OnServerReady`	Fires when a backend server has finished registering with the load balancer and is ready to accept traffic.

Quick Start

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

> **About this scenario.** Fires before the load balancer sends a backend server binding (host, port, protocol) to a newly accepted downstream client.

Delphi (VCL / FireMonkey)

```
procedure OnBeforeSendServerBinding(Connection: TsgcWSConnection;
  var Binding: TsgcWSLoadBalancerServerBinding);
begin
  // force secure WebSocket scheme when the client connected over TLS
  if Connection.IsSSL then
    Binding.Protocol := 'wss';
end;
```

C++ Builder

```
void OnBeforeSendServerBinding(TsgcWSConnection *Connection,
  TsgcWSLoadBalancerServerBinding *&Binding)
{
  if (Connection->IsSSL)
    Binding->Protocol = "wss";
}
```

.NET (C#)

```
void OnBeforeSendServerBinding(TsgcWSConnection Connection,
  out TsgcWSLoadBalancerServerBinding Binding)
{
  if (Connection.IsSSL)
    Binding.Protocol = "wss";
}
```

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 · Broadcast

Fans a WebSocket message out across every backup server in the cluster, optionally filtered by channel, protocol, or connection GUID list.

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

```
oServer.Broadcast('Hello From Server');
```

```
C++ Builder
```

```
oServer->Broadcast("Hello From Server");
```

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

```
oServer.Broadcast("Hello From Server");
```

2 · DisconnectAll

Disconnects every active client connection and every registered backup server while keeping the load balancer listening for new connections.

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

```
oServer.DisconnectAll;
```

```
C++ Builder
```

```
oServer->DisconnectAll();
```

.NET (C#)

```
oServer.DisconnectAll();
```

3 · OnBinary

Fires when the load balancer itself receives a binary WebSocket frame on one of its downstream sessions.

Delphi (VCL / FireMonkey)

```
procedure OnBinary(Connection: TsgcWSConnection; const Data: TMemoryStream);
begin
  Log(Format('Received %d bytes from %s', [Data.Size, Connection.Guid]));
end;
```

C++ Builder

```
void OnBinary(TsgcWSConnection *Connection, const TMemoryStream *Data)
{
  Log("Received " + IntToStr(Data->Size) + " bytes from " + Connection->Guid);
}
```

.NET (C#)

```
void OnBinary(TsgcWSConnection Connection, TMemoryStream Data)
{
  Console.WriteLine("Received " + Data.Size + " bytes from " + Connection.Guid);
}
```

4 · OnClientBinary

Fires when a binary frame is received from a downstream client before it is forwarded to the selected backend server.

Delphi (VCL / FireMonkey)

```
procedure OnClientBinary(Connection: TsgcWSConnection; Data: TMemoryStream;
  var Handled: Boolean);
begin
  // drop empty frames instead of forwarding them
  Handled := Data.Size = 0;
end;
```

C++ Builder

```
void OnClientBinary(TsgcWSConnection *Connection, TMemoryStream *Data,
  bool &Handled)
{
  Handled = (Data->Size == 0);
}
```

.NET (C#)

```
void OnClientBinary(TsgcWSConnection Connection, TMemoryStream Data,
  out bool Handled)
{
  Handled = (Data.Size == 0);
}
```

5 · OnClientConnect

Fires when a downstream client finishes the WebSocket handshake against the load balancer and is paired with a backend server.

Delphi (VCL / FireMonkey)

```
procedure OnClientConnect(ServerConnection: TsgcWSConnection;
  ClientConnection: TsgcWSLoadBalancerClientConnection);
begin
  Log(Format('Client %s routed to backend %s',
    [ClientConnection.Guid, ServerConnection.Guid]));
end;
```

C++ Builder

```
void OnClientConnect(TsgcWSConnection *ServerConnection,
  TsgcWSLoadBalancerClientConnection *ClientConnection)
{
  Log("Client " + ClientConnection→Guid
    + " routed to backend " + ServerConnection→Guid);
}
```

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

```
void OnClientConnect(TsgcWSConnection ServerConnection,
  TsgcWSLoadBalancerClientConnection ClientConnection)
{
  Console.WriteLine("Client " + ClientConnection.Guid
    + " routed to backend " + ServerConnection.Guid);
}
```

6 · OnClientDisconnect

Fires when a downstream client session is dropped from the load balancer, either because the client left or because the backend terminated.

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

```
procedure OnClientDisconnect(ServerConnection: TsgcWSConnection;
  ClientConnection: TsgcWSLoadBalancerClientConnection);
begin
  Log('Client disconnected: ' + ClientConnection.Guid);
end;
```

```
C++ Builder
```

```
void OnClientDisconnect(TsgcWSConnection *ServerConnection,
  TsgcWSLoadBalancerClientConnection *ClientConnection)
{
  Log("Client disconnected: " + ClientConnection→Guid);
}
```

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

```
void OnClientDisconnect(TsgcWSConnection ServerConnection,
  TsgcWSLoadBalancerClientConnection ClientConnection)
{
  Console.WriteLine("Client disconnected: " + ClientConnection.Guid);
}
```

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.

Online help — component page	www.esegece.com/help/sgcWebSockets/Components/TsgcWebSocketLoadBalancerServer.htm
Delphi demo project (in the sgcWebSockets package)	`Demos\04.WebSocket_Other_Samples\03.LoadBalancer_Server`
Component page	www.esegece.com/products/websockets/components/scaling/load-balancer/
Product page	www.esegece.com/products/websockets/

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