SGCWEBSOCKETS · TECHNICAL DOCUMENT

# Rate Limiter

Per-connection and per-key rate-limiting component — token bucket and fixed-window strategies for any sgcWebSockets server.

eSeGeCe

WWW.ESEGECE.COM

## Overview

TsgcWSRateLimiter implements a full-featured rate limiting component that protects server endpoints from excessive traffic, abuse and scraping.

## At a glance

| | |
|---|---|
| COMPONENT CLASS<br>`TsgcWSRateLimiter` | STANDARDS / SPEC<br>— |
| TRANSPORTS<br>TCP, TLS | PLATFORMS<br>Windows, macOS, Linux, iOS, Android |
| FRAMEWORKS<br>VCL, FireMonkey, Lazarus / FPC | EDITION<br>Standard / Professional / Enterprise |

## Features

- Native Delphi implementation with full ANSI/Unicode support.

# Technical specification

| | |
|---|---|
| **Component class** | `TsgcWSRateLimiter` (unit `sgcWebSocket_RateLimiter`) |
| **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.

| | |
|---|---|
| `PerAPIKey` | Rate-limit rule scoped to an API-key identifier (keys prefixed apikey:). |
| `TokenBucket` | Token Bucket algorithm configuration: capacity plus refill rate for smooth bursts. |
| `PerEndpoint` | Collection of pattern-based rules matched by wildcard against the request key. |
| `Enabled` | Master switch that turns the entire rate limiter on or off. |
| `SlidingWindow` | Sliding Window algorithm configuration: MaxRequests over a rolling WindowSec. |
| `FixedWindow` | Fixed Window algorithm configuration: MaxRequests inside a clock-aligned WindowSec. |
| `BurstProtection` | Short-timescale spike detector that puts abusive keys into a cooldown. |
| `PerIP` | Rate-limit rule scoped to the source IP address. |
| `PerUser` | Rate-limit rule scoped to a user identifier (keys prefixed user:). |
| `Quotas` | Long-term caps (hour/day/month) evaluated on top of the main strategy. |

## Main methods

The principal public methods exposed by the component.

| | |
|---|---|
| `IsConnectionAllowed()` | Server hook: returns True if a new connection from IP passes rate limiting. |
| `RegisterConnection()` | Registers a new connection for tracking (called automatically by the server). |
| `UnregisterConnection()` | Removes a connection from tracking (called automatically on disconnect). |
| `Reset()` | Clears the counters for a single key. |
| `ResetAll()` | Clears every internal counter and resets the Stats object. |
| `SaveStateToStream()` | Persists the internal state to an arbitrary TStream. |
| `LoadStateFromStream()` | Restores the internal state from an arbitrary TStream. |
| `IsAllowed()` | Returns True if a request of the given cost is allowed for the specified key. |
| `Consume()` | Consumes cost tokens and returns a detailed TsgcRateLimitResult. |
| `IsMessageAllowed()` | Server hook: returns True if a message from IP passes rate limiting. |

## Public events

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

| | |
|---|---|
| `OnQuotaExceeded` | property OnQuotaExceeded: TsgcRateLimitOnQuotaExceeded; // TsgcRateLimitOnQuotaExceeded = procedure(Sender: TObject; const aKey, aQuotaName: string) of object __property TsgcRateLimitOnQuotaExceeded O... |
| `OnStateChange` | property OnStateChange: TsgcRateLimitOnStateChange; // TsgcRateLimitOnStateChange = procedure(Sender: TObject; const aKey: string; aOldBucketSize, aNewBucketSize: Integer) of object __property TsgcRat... |
| `OnThrottled` | Fired when a request is rejected by the strategy or burst protection. |

## Quick Start

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

> **About this scenario.** Server hook: returns True if a new connection from IP passes rate limiting.

### Delphi (VCL / FireMonkey)

```
// Manual gate in a custom OnConnect handler
procedure TForm1.ServerConnect(Connection: TsgcWSConnection);
begin
  if not sgcWSRateLimiter1.IsConnectionAllowed(Connection.PeerIP) then
    Connection.Disconnect;
end;
```

## 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 · RegisterConnection

Registers a new connection for tracking (called automatically by the server).

Delphi (VCL / FireMonkey)

```
procedure TForm1.ServerConnect(Connection: TsgcWSConnection);
begin
  sgcWSRateLimiter1.RegisterConnection(Connection.PeerIP);
end;
```

### 2 · UnregisterConnection

Removes a connection from tracking (called automatically on disconnect).

Delphi (VCL / FireMonkey)

```
procedure TForm1.ServerDisconnect(Connection: TsgcWSConnection);
begin
  sgcWSRateLimiter1.UnregisterConnection(Connection.PeerIP);
end;
```

### 3 · BurstProtection

Short-timescale spike detector that puts abusive keys into a cooldown.

Delphi (VCL / FireMonkey)

```
// More than 50 requests in 500ms triggers a 30-second cooldown
sgcWSRateLimiter1.BurstProtection.Enabled := True;
sgcWSRateLimiter1.BurstProtection.BurstThreshold := 50;
sgcWSRateLimiter1.BurstProtection.BurstWindowMs := 500;
sgcWSRateLimiter1.BurstProtection.CooldownSec := 30;
```

## 4 · Consume

Consumes cost tokens and returns a detailed TsgcRateLimitResult.

Delphi (VCL / FireMonkey)

```
var
  oResult: TsgcRateLimitResult;
begin
  oResult := sgcWSRateLimiter1.Consume('apikey:' + ApiKey);
  if not oResult.Allowed then
  begin
    Response.StatusCode := 429;
    Response.CustomHeaders.Values['Retry-After'] := IntToStr(oResult.RetryAfterSec);
    Response.ContentText := 'Throttled: ' + oResult.Reason;
    Exit;
  end;
end;
```

## 5 · Enabled

Master switch that turns the entire rate limiter on or off.

Delphi (VCL / FireMonkey)

```
// Temporarily disable rate limiting during a load test
sgcWSRateLimiter1.Enabled := False;
try
  RunLoadTest;
finally
  sgcWSRateLimiter1.Enabled := True;
end;
```

## 6 · FixedWindow

Fixed Window algorithm configuration: MaxRequests inside a clock-aligned WindowSec.

Delphi (VCL / FireMonkey)

```
// Billing-style: 1000 requests per hour, resets on the hour
sgcWSRateLimiter1.FixedWindow.Enabled := True;
sgcWSRateLimiter1.FixedWindow.WindowSec := 3600;
sgcWSRateLimiter1.FixedWindow.MaxRequests := 1000;
```

## 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/TsgcWSRateLimiter.htm |
| **Delphi demo project (in the sgcWebSockets package)** | `Demos\04.WebSocket_Other_Samples\14.RateLimiter` |
| **Component page** | www.esegece.com/products/websockets/components/resilience/rate-limiter/ |
| **Product page** | www.esegece.com/products/websockets/ |

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