SGCWEBSOCKETS · TECHNICAL DOCUMENT

Circuit Breaker

Circuit-breaker resilience pattern for sgcWebSockets — automatic open/half-open/closed transitions wrap any HTTP or WebSocket call.

eSeGeCe

WWW.ESEGECE.COM

Overview

TsgcWSCircuitBreaker implements the Circuit Breaker resilience pattern for client-side calls to HTTP APIs.

At a glance

COMPONENT CLASS `TsgcWSCircuitBreaker`	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	`TsgcWSCircuitBreaker` (unit `sgcWebSocket_CircuitBreaker`)
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.

`ServerKey`	Key used by the server-side integration hooks (IsConnectionAllowed, IsMessageAllowed, RecordMessageError, RecordMessageSuccess).
`PerEndpoint`	Collection of pattern-based overrides that apply different Thresholds/Recovery per host.
`Enabled`	Master switch that turns the whole circuit breaker on or off.
`DefaultKey`	Key used by the parameterless overloads of Execute, RecordSuccess and RecordFailure.
`Thresholds`	Conditions that trip the circuit from Closed to Open (failure counts, failure rate and slow-call rate).
`TimeWindow`	Rolling window (width and bucket count) against which all thresholds are evaluated.
`Recovery`	Half-open retry policy that governs how an Open circuit transitions back to Closed.
`Fallback`	Alternative payload returned through OnFallback while the circuit is Open.
`Classification`	Rules that decide which exceptions are recorded as failures and which are ignored.
`Metrics`	Read-only aggregate counters exposed for dashboards, logging and alerting.

Main methods

The principal public methods exposed by the component.

`IsCallAllowed()`	Returns True when a new call is currently allowed for the given key; advances HalfOpen trial accounting.
`IsConnectionAllowed()`	Server-side gatekeeper that returns False while the ServerKey circuit is Open.
`RegisterConnection()`	Server-side hook that tracks a new connection (reserved for future per-IP metrics).
`UnregisterConnection()`	Server-side hook that stops tracking a disconnected connection (counterpart of RegisterConnection).
`ForceOpen()`	Manually moves the circuit for the given key into the Open state.
`ForceClose()`	Manually moves the circuit for the given key back to the Closed state.
`Reset()`	Clears the state, rolling-window counters and last-success payload for a single key.
`ResetAll()`	Clears every tracked circuit, every rolling-window counter and all aggregate Metrics.
`Execute()`	Runs a protected action; checks IsCallAllowed first and records success / failure / slow automatically.
`ExecuteWithResult()`	Runs a protected action that returns a TObject and reports the result via an out parameter.

Public events

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

`OnCallRejected`	Fired when the breaker refuses a call because the circuit is Open or the HalfOpen trial budget is exhausted.
`OnFailureRecorded`	Fired every time a failure is recorded against a circuit, after Classification has accepted it.
`OnFallback`	Fired just before a fallback response is returned while the circuit is Open; the handler may replace the payload.

`OnSlowCall`	Fired when a successful call exceeds Thresholds.SlowCallDurationMs.
`OnStateChange`	Fired when a circuit transitions between Closed, Open and HalfOpen.

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-side gatekeeper that returns False while the ServerKey circuit is Open.

Delphi (VCL / FireMonkey)

```
procedure TForm1.WSServerConnect(Connection: TsgcWSConnection);
begin
  if not sgcWSCircuitBreaker1.IsConnectionAllowed(Connection.PeerIP) then
  begin
    Connection.Disconnect;
    Exit;
  end;
  sgcWSCircuitBreaker1.RegisterConnection(Connection.PeerIP);
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

Server-side hook that tracks a new connection (reserved for future per-IP metrics).

Delphi (VCL / FireMonkey)

```
procedure TForm1.WSServerConnect(Connection: TsgcWSConnection);
begin
  if not sgcWSCircuitBreaker1.IsConnectionAllowed(Connection.PeerIP) then
  begin
    Connection.Disconnect;
    Exit;
  end;
  sgcWSCircuitBreaker1.RegisterConnection(Connection.PeerIP);
end;

procedure TForm1.WSServerDisconnect(Connection: TsgcWSConnection);
begin
  sgcWSCircuitBreaker1.UnregisterConnection(Connection.PeerIP);
end;
```

2 · UnregisterConnection

Server-side hook that stops tracking a disconnected connection (counterpart of RegisterConnection).

Delphi (VCL / FireMonkey)

```
procedure TForm1.WSServerDisconnect(Connection: TsgcWSConnection);
begin
  sgcWSCircuitBreaker1.UnregisterConnection(Connection.PeerIP);
end;
```

3 · Classification

Rules that decide which exceptions are recorded as failures and which are ignored.

Delphi (VCL / FireMonkey)

```
// Record HTTP 5xx and timeouts as failures; ignore client-side 4xx
sgcWSCircuitBreaker1.Classification.MatchMode := cemContains;
sgcWSCircuitBreaker1.Classification.RecordAsFailure.Add('HTTP/1.1 500');
sgcWSCircuitBreaker1.Classification.RecordAsFailure.Add('HTTP/1.1 502');
sgcWSCircuitBreaker1.Classification.RecordAsFailure.Add('HTTP/1.1 503');
sgcWSCircuitBreaker1.Classification.RecordAsFailure.Add('HTTP/1.1 504');
sgcWSCircuitBreaker1.Classification.RecordAsFailure.Add('timeout');
sgcWSCircuitBreaker1.Classification.IgnoreExceptions.Add('HTTP/1.1 400');
sgcWSCircuitBreaker1.Classification.IgnoreExceptions.Add('HTTP/1.1 401');
sgcWSCircuitBreaker1.Classification.IgnoreExceptions.Add('HTTP/1.1 404');
```

4 · DefaultKey

Key used by the parameterless overloads of Execute, RecordSuccess and RecordFailure.

Delphi (VCL / FireMonkey)

```
// Use a meaningful key so Metrics and events report a readable label
sgcWSCircuitBreaker1.DefaultKey := 'openai-api';

// Now the parameterless overloads target the 'openai-api' circuit
sgcWSCircuitBreaker1.Execute(
  procedure
  begin
    CallOpenAI;
  end);
```

5 · Enabled

Master switch that turns the whole circuit breaker on or off.

Delphi (VCL / FireMonkey)

```
// Temporarily disable the breaker during a load test
sgcWSCircuitBreaker1.Enabled := False;
try
  RunLoadTest;
finally
  sgcWSCircuitBreaker1.Enabled := True;
end;
```

6 · Execute

Runs a protected action; checks IsCallAllowed first and records success / failure / slow automatically.

Delphi (VCL / FireMonkey)

```
// Run a protected call against OpenAI under key 'openai-api'
if not sgcWSCircuitBreaker1.Execute('openai-api',
  procedure
  begin
    CallOpenAIEndpoint;
  end) then
  LogRejected('Circuit open for openai-api');
```

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/TsgcWSCircuitBreaker.htm
Delphi demo project (in the sgcWebSockets package)	`Demos\04.WebSocket_Other_Samples\15.CircuitBreaker`
Component page	www.esegece.com/products/websockets/components/resilience/circuit-breaker/
Product page	www.esegece.com/products/websockets/

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