SGCWEBSOCKETS · TECHNICAL DOCUMENT

OAuth2 Server

TsgcHTTP_OAuth2_Server — OAuth 2.0 authorisation-server implementation for sgcWebSockets HTTP / WebSocket servers.

eSeGeCe

WWW.ESEGECE.COM

Overview

This component provides the OAuth2 protocol implementation in Server Side Components.

At a glance

COMPONENT CLASS

`TsgcHTTP_OAuth2_Server`

STANDARDS / SPEC

OAuth 2.0 — RFC 6749

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	OAuth 2.0 — RFC 6749 · OAuth 2.0 Bearer Tokens — RFC 6750
Component class	`TsgcHTTP_OAuth2_Server` (unit `sgcHTTP_OAuth2_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.

`OAuth2Options`	Authorization-server configuration: endpoint URLs, token lifetimes, PKCE, DPoP, revocation, introspection and device-code settings.
`Version`	Read-only string exposing the sgcWebSockets library version.

Main methods

The principal public methods exposed by the component.

`IsOAuth2TokenValid()`	Validates an incoming Bearer access token presented by the client, either by parsing the request headers or by taking the raw token string.
`IsOAuth2Unauthorized()`	Checks whether a request on the given connection must be rejected with an OAuth 2.0 unauthorized response.
`AddToken()`	Pre-seeds an access/refresh token pair into the server token store for a registered application.
`RemoveToken()`	Revokes a specific access token previously issued by the server.
`RemoveTokenByRefreshToken()`	Revokes the token pair identified by its refresh token value.
`RegisterApp()`	Registers a new OAuth 2.0 client application on the server and returns its generated credentials.

`UnRegisterApp()`	Removes a previously registered OAuth 2.0 client application from the server.
`RegisterProvider()`	Registers an external identity provider (Google, GitHub, Microsoft, etc.) so the server can delegate federated sign-in to it.
`UnRegisterProvider()`	Removes a previously registered external identity provider from the server.
`ClearProviders()`	Removes every external identity provider registered in the server federation list.

Public events

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

`OnOAuth2AfterAccessToken`	TsgcHTTP_OAuth2_Server › Events › OnOAuth2AfterAccessToken
`OnOAuth2AfterIntrospectToken`	TsgcHTTP_OAuth2_Server › Events › OnOAuth2AfterIntrospectToken
`OnOAuth2AfterRefreshToken`	TsgcHTTP_OAuth2_Server › Events › OnOAuth2AfterRefreshToken
`OnOAuth2AfterRevokeToken`	TsgcHTTP_OAuth2_Server › Events › OnOAuth2AfterRevokeToken
`OnOAuth2AfterValidateAccessToken`	TsgcHTTP_OAuth2_Server › Events › OnOAuth2AfterValidateAccessToken
`OnOAuth2Authentication`	Validates user credentials submitted on the sign-in page during the authorize flow.
`OnOAuth2BeforeDispatchPage`	TsgcHTTP_OAuth2_Server › Events › OnOAuth2BeforeDispatchPage
`OnOAuth2BeforeRequest`	Fires before any OAuth2 endpoint processes an incoming HTTP request; lets the application inspect, rewrite or cancel the request.
`OnOAuth2DeviceAuthorization`	TsgcHTTP_OAuth2_Server › Events › OnOAuth2DeviceAuthorization

`OnOAuth2DeviceCodeVerification`	TsgcHTTP_OAuth2_Server › Events › OnOAuth2DeviceCodeVerification
`OnOAuth2ResponseError`	Fires when the server is about to return an OAuth2 error response (invalid_grant, invalid_client, access_denied, etc.) so the application can override status, body or headers.
`OnOAuth2Unauthorized`	Fires when a protected endpoint rejects a request because the bearer token is missing, invalid or expired.
`OnOAuth2ValidateDPoP`	Fires when a resource request carries a DPoP proof header so the application can verify the proof-of-possession (RFC 9449).

Quick Start

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

> **About this scenario.** The following example shows a complete server setup with the Authorization Code grant, including app registration and event handlers.

Delphi (VCL / FireMonkey)

```
// Create and configure OAuth2 server
OAuth2Server := TsgcHTTP_OAuth2_Server.Create(nil);

// Register an application
OAuth2Server.Apps.AddApp('MyApp', 'http://127.0.0.1:8080',
  'my-client-id', 'my-client-secret', 3600, True, [auth2Code]);

// Attach to HTTP server
Server.Authentication.Enabled := True;
Server.Authentication.OAuth.OAuth2 := OAuth2Server;

// Handle authentication
procedure TForm1.OAuth2ServerOAuth2Authentication(Sender: TObject;
  const User, Password: String; var Authenticated: Boolean);
begin
  Authenticated := (User = 'admin') and (Password = 'secret');
end;

// Handle token issuance
procedure TForm1.OAuth2ServerOAuth2AfterAccessToken(Sender: TObject;
  const Token, RefreshToken: String; ExpiresIn: Integer);
begin
  Log('Access token issued: ' + Token);
end;
```

C++ Builder

```
// Create and configure OAuth2 server
TsgcHTTP_OAuth2_Server *OAuth2Server = new TsgcHTTP_OAuth2_Server(this);

// Register an application
OAuth2Server->Apps->AddApp("MyApp", "http://127.0.0.1:8080",
  "my-client-id", "my-client-secret", 3600, true,
  TsgcOAuth2GrantTypes() << auth2Code);

// Attach to HTTP server
Server->Authentication->Enabled = true;
Server->Authentication->OAuth->OAuth2 = OAuth2Server;

void __fastcall TForm1::OAuth2ServerOAuth2Authentication(TObject *Sender,
  const UnicodeString User, const UnicodeString Password, bool &Authenticated)
{
  Authenticated = (User == "admin") && (Password == "secret");
}

void __fastcall TForm1::OAuth2ServerOAuth2AfterAccessToken(TObject *Sender,
  const UnicodeString Token, const UnicodeString RefreshToken, int ExpiresIn)
{
  Log("Access token issued: " + Token);
}
```

.NET (C#)

```
// Create and configure OAuth2 server
var OAuth2Server = new TsgcHTTP_OAuth2_Server();

// Register an application
OAuth2Server.Apps.AddApp("MyApp", "http://127.0.0.1:8080",
  "my-client-id", "my-client-secret", 3600, true,
  new TsgcOAuth2GrantType[] { TsgcOAuth2GrantType.auth2Code });

// Attach to HTTP server
Server.Authentication.Enabled = true;
Server.Authentication.OAuth.OAuth2 = OAuth2Server;

OAuth2Server.OnOAuth2Authentication += (sender, user, password, ref authenticated) =>
{
  authenticated = (user == "admin") && (password == "secret");
};

OAuth2Server.OnOAuth2AfterAccessToken += (sender, token, refreshToken, expiresIn) =>
{
  Console.WriteLine("Access token issued: " + token);
};
```

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 · OAuth2 | None Authenticate URLs

By default, when OAuth2 is enabled on Server Side, all the HTTP Requests require Authentication using Bearer Tokens.

Delphi (VCL / FireMonkey)

```
procedure OnOAuth2BeforeRequest(Sender: TObject; aConnection: TsgcWSConnection; aHeaders: TStrin
 var Cancel: Boolean);
begin
if DecodeGETFullPath(aHeaders) = '/Public.html' then
Cancel := True
else if DecodePOSTFullPath(aHeaders) = '/Form.html' then
Cancel := True;
end;
```

C++ Builder

```
procedure OnOAuth2BeforeRequest(TObject *Sender; TsgcWSConnection *aConnection; TStringList *aHe
 ref bool Cancel)
{
if (DecodeGETFullPath(aHeaders) == "/Public.html")
{
Cancel = true;
}
}
```

.NET (C#)

```
procedure OnOAuth2BeforeRequest(TObject Sender; TsgcWSConnection aConnection; TStringList aHeade
  ref bool Cancel)
{
if (DecodeGETFullPath(aHeaders) == "/Public.html")
{
Cancel = true;
}
}
```

2 · OAuth2 | Server Authentication

When an OAuth2 client requests a new Authorization, the server shows a web page where the user must allow the connection and then login. This page is provided by sgcWebSockets library and is dispatched automatically when a client requests an Authorization.

Delphi (VCL / FireMonkey)

```
procedure OnAuth2Authentication(Connection: TsgcWSConnection; OAuth2: TsgcHTTPOAuth2Request; aUs
 aPassword: string; var Authenticated: Boolean);
begin
if ((aUser = 'user') and (aPassword = 'secret')) then
Authenticated := True;
end;
```

C++ Builder

```
void OnOAuth2Authentication(TsgcWSConnection *Connection, TsgcHTTPOAuth2Request *OAuth2, string
 string aPassword, ref bool Authenticated)
{
if ((aUser == "user") and (aPassword == "secret"))
{
Authenticated = true;
}
}
```

.NET (C#)

```
void OnOAuth2Authentication(TsgcWSConnection Connection, TsgcHTTPOAuth2Request OAuth2, string aU
  string aPassword, ref bool Authenticated)
{
if ((aUser == "user") and (aPassword == "secret"))
{
Authenticated = true;
}
}
```

3 · App Registration

Before clients can use the Authorization Code flow, you must register the application on the server. Use Apps.AddApp and ensure the AllowedGrantTypes includes auth2Code.

Delphi (VCL / FireMonkey)

```
OAuth2Server.Apps.AddApp('MyApp', 'http://127.0.0.1:8080',
  'my-client-id', 'my-client-secret', 3600, True, [auth2Code]);
```

C++ Builder

```
OAuth2Server->Apps->AddApp("MyApp", "http://127.0.0.1:8080",
  "my-client-id", "my-client-secret", 3600, true,
  TsgcOAuth2GrantTypes() << auth2Code);
```

.NET (C#)

```
OAuth2Server.Apps.AddApp("MyApp", "http://127.0.0.1:8080",
  "my-client-id", "my-client-secret", 3600, true,
  new TsgcOAuth2GrantType[] { TsgcOAuth2GrantType.auth2Code });
```

4 · DPoP Validation (RFC 9449)

DPoP (Demonstrating Proof of Possession) per RFC 9449 enables the TsgcHTTP_OAuth2_Server to require sender-constrained tokens. When DPoP is enabled, the server validates DPoP proof JWTs, binds tokens to the client's JWK thumbprint, and returns token_type: DPoP instead of Bearer.

Delphi (VCL / FireMonkey)

```
OAuth2Server := TsgcHTTP_OAuth2_Server.Create(nil);
// Enable DPoP
OAuth2Server.OAuth2Options.DPoP := True;
// Register app
OAuth2Server.Apps.AddApp('MyApp', 'http://127.0.0.1:8080',
  'my-client-id', 'my-client-secret', 3600, True, [auth2Code]);
Server.Authentication.Enabled := True;
Server.Authentication.OAuth.OAuth2 := OAuth2Server;
// Custom DPoP validation
procedure TForm1.OAuth2ServerOAuth2ValidateDPoP(Sender: TObject;
  const DPoPProof, AccessToken, Method, URL: String; var Valid: Boolean);
begin
  // Additional custom checks
  Log('DPoP proof validated for: ' + Method + ' ' + URL);
  Valid := True; // Set to False to reject the request
end;
```

C++ Builder

```
TsgcHTTP_OAuth2_Server *OAuth2Server = new TsgcHTTP_OAuth2_Server(this);
OAuth2Server->OAuth2Options->DPoP = true;
OAuth2Server->Apps->AddApp("MyApp", "http://127.0.0.1:8080",
  "my-client-id", "my-client-secret", 3600, true,
  TsgcOAuth2GrantTypes() << auth2Code);
Server->Authentication->Enabled = true;
Server->Authentication->OAuth->OAuth2 = OAuth2Server;
void __fastcall TForm1::OAuth2ServerOAuth2ValidateDPoP(TObject *Sender,
  const UnicodeString DPoPProof, const UnicodeString AccessToken,
  const UnicodeString Method, const UnicodeString URL, bool &Valid)
{
  Log("DPoP proof validated for: " + Method + " " + URL);
  Valid = true;
}
```

.NET (C#)

```
var OAuth2Server = new TsgcHTTP_OAuth2_Server();
OAuth2Server.OAuth2Options.DPoP = true;
OAuth2Server.Apps.AddApp("MyApp", "http://127.0.0.1:8080",
  "my-client-id", "my-client-secret", 3600, true,
  new TsgcOAuth2GrantType[] { TsgcOAuth2GrantType.auth2Code });
Server.Authentication.Enabled = true;
Server.Authentication.OAuth.OAuth2 = OAuth2Server;
OAuth2Server.OnOAuth2ValidateDPoP += (sender, dpopProof, accessToken, method, url, ref valid) =>
{
  Console.WriteLine("DPoP proof validated for: " + method + " " + url);
  valid = true;
};
```

5 · Device Authorization Grant (RFC 8628)

The Device Authorization Grant is designed for input-constrained devices (smart TVs, IoT devices, CLI tools) that cannot easily display a browser or accept keyboard input. The TsgcHTTP_OAuth2_Server provides a device authorization endpoint and a verification page so that users can authorize the device from a secondary device such as a phone or computer.

Delphi (VCL / FireMonkey)

```
OAuth2Server := TsgcHTTP_OAuth2_Server.Create(nil);
// Enable device authorization
OAuth2Server.OAuth2Options.DeviceAuthorization.Enabled := True;
OAuth2Server.OAuth2Options.DeviceAuthorization.ExpiresIn := 600;
OAuth2Server.OAuth2Options.DeviceAuthorization.Interval := 5;
// Register app
OAuth2Server.Apps.AddApp('DeviceApp', '',
  'device-client-id', 'device-client-secret', 3600, True,
  [auth2Code]);
Server.Authentication.Enabled := True;
Server.Authentication.OAuth.OAuth2 := OAuth2Server;
// Handle device code verification
procedure TForm1.OAuth2ServerOAuth2DeviceCodeVerification(Sender: TObject;
  const UserCode: String; var Authorized: Boolean);
begin
  // Validate the user code and authorize the device
  Authorized := True;
end;
procedure TForm1.OAuth2ServerOAuth2AfterAccessToken(Sender: TObject;
  const Token, RefreshToken: String; ExpiresIn: Integer);
begin
  Log('Device authorized. Token: ' + Token);
end;
```

C++ Builder

```
TsgcHTTP_OAuth2_Server *OAuth2Server = new TsgcHTTP_OAuth2_Server(this);
OAuth2Server->OAuth2Options->DeviceAuthorization->Enabled = true;
OAuth2Server->OAuth2Options->DeviceAuthorization->ExpiresIn = 600;
OAuth2Server->OAuth2Options->DeviceAuthorization->Interval = 5;
OAuth2Server->Apps->AddApp("DeviceApp", "",
  "device-client-id", "device-client-secret", 3600, true,
  TsgcOAuth2GrantTypes() << auth2Code);
Server->Authentication->Enabled = true;
Server->Authentication->OAuth->OAuth2 = OAuth2Server;
void __fastcall TForm1::OAuth2ServerOAuth2DeviceCodeVerification(TObject *Sender,
  const UnicodeString UserCode, bool &Authorized)
{
  Authorized = true;
}
void __fastcall TForm1::OAuth2ServerOAuth2AfterAccessToken(TObject *Sender,
  const UnicodeString Token, const UnicodeString RefreshToken, int ExpiresIn)
{
  Log("Device authorized. Token: " + Token);
}
```

.NET (C#)

```
var OAuth2Server = new TsgcHTTP_OAuth2_Server();
OAuth2Server.OAuth2Options.DeviceAuthorization.Enabled = true;
OAuth2Server.OAuth2Options.DeviceAuthorization.ExpiresIn = 600;
OAuth2Server.OAuth2Options.DeviceAuthorization.Interval = 5;
OAuth2Server.Apps.AddApp("DeviceApp", "",
  "device-client-id", "device-client-secret", 3600, true,
  new TsgcOAuth2GrantType[] { TsgcOAuth2GrantType.auth2Code });
Server.Authentication.Enabled = true;
Server.Authentication.OAuth.OAuth2 = OAuth2Server;
OAuth2Server.OnOAuth2DeviceCodeVerification += (sender, userCode, ref authorized) =>
{
  authorized = true;
};
OAuth2Server.OnOAuth2AfterAccessToken += (sender, token, refreshToken, expiresIn) =>
{
  Console.WriteLine("Device authorized. Token: " + token);
};
```

6 · OAuth2 | Customize Sign-In HTML

When an OAuth2 client do a request to get a new Access Token, a Web-Page is shown in a web-browser to Allow this connection and login with an User and Password.

Delphi (VCL / FireMonkey)

```
procedure OnOAuth2BeforeDispatchPage(Sender: TObject; OAuth2: TsgcHTTPOAuth2Request; var HTML: s
begin
HTML := 'your custom html';
end;
```

C++ Builder

```
void OnOAuth2BeforeDispatchPage(TObject *Sender; TsgcHTTPOAuth2Request *OAuth2; ref string HTML)
{
HTML = "your custom html";
}
```

.NET (C#)

```
void OnOAuth2BeforeDispatchPage(TObject Sender; TsgcHTTPOAuth2Request OAuth2; ref string HTML)
{
HTML = "your custom html";
}
```

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 — OAuth 2.0 — RFC 6749	datatracker.ietf.org/doc/html/rfc6749
Primary standard / spec — OAuth 2.0 Bearer Tokens — RFC 6750	datatracker.ietf.org/doc/html/rfc6750
Online help — component page	www.esegece.com/help/sgcWebSockets/Components/HTTP/Authorization/OAuth2/server/TsgcHTTP_OAuth2_Server.htm
Delphi demo project (in the sgcWebSockets package)	`Demos\20.HTTP_Protocol\08.OAuth2_ServerProvider`
Component page	www.esegece.com/products/websockets/http/oauth2-server/
Product page	www.esegece.com/products/websockets/

> **Document scope.** This document covers the publicly-documented surface of the OAuth2 Server component shipped with sgcWebSockets. For full property, method and event reference consult the online help linked above.