SGCWEBSOCKETS · TECHNICAL DOCUMENT

OAuth2 Client

TsgcHTTP_OAuth2_Client — OAuth 2.0 client with authorisation-code, PKCE, client-credentials, device-code and refresh-token flows.

eSeGeCe

WWW.ESEGECE.COM

Overview

When a client needs a new Access Token, automatically starts an HTTP server to process response from Authorization server.

At a glance

COMPONENT CLASS `TsgcHTTP_OAuth2_Client`	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 · PKCE — RFC 7636
Component class	`TsgcHTTP_OAuth2_Client` (unit `sgcHTTP_OAuth2_Client`)
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.

`AuthorizationServerOptions`	Endpoints published by the external OAuth 2.0 authorization server the client talks to.
`HTTPClientOptions`	TLS and logging configuration of the internal HTTP client used to POST to the token, revocation and introspection endpoints.
`LocalServerOptions`	Configuration for the local HTTP listener that receives the authorization-code redirect from the browser.
`OAuth2Options`	Client identity and grant-type selection driving the OAuth 2.0 flow.
`DPoPOptions`	DPoP (Demonstrating Proof-of-Possession, RFC 9449) key material and signing options attached to token requests and protected resource calls.
`Version`	Read-only string exposing the sgcWebSockets library version.

Main methods

The principal public methods exposed by the component.

`Start()`	Starts the configured OAuth2 flow to obtain an access token.
`Stop()`	Aborts any pending OAuth2 flow and shuts the local redirect server.
`Refresh()`	Requests a new access token using a refresh token.

`GenerateDPoPKeyPair()`	Creates a DPoP-compliant key pair and populates DPoPOptions.
`Revoke()`	Revokes an access or refresh token per RFC 7009.
`Introspect()`	Queries token status and metadata per RFC 7662.
`PublicKeyToJWK()`	Converts a PEM-encoded public key to its JWK representation.

Public events

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

`OnAfterAccessToken`	Fires when the token endpoint returns a successful access-token response.
`OnAfterAuthorizeCode`	Fires when the authorization server redirects back with the authorization code.
`OnAfterIntrospectToken`	Fires after the introspection endpoint returns the token metadata.
`OnAfterRefreshToken`	Fires when the token endpoint returns a new access token from a refresh_token grant.
`OnAfterRevokeToken`	Fires after the revocation endpoint successfully invalidates the token.
`OnBeforeAccessToken`	Fires before the client posts to the token endpoint to exchange the code for an access token.
`OnBeforeAuthorizeCode`	Fires before the client opens the browser to request user authorization.
`OnBeforeIntrospectToken`	TsgcHTTP_OAuth2_Client › Events › OnBeforeIntrospectToken
`OnBeforeRefreshToken`	Fires before the client posts to the token endpoint to redeem the refresh token.
`OnBeforeRevokeToken`	Fires before the client posts to the revocation endpoint (RFC 7009).
`OnDPoPSign`	Fires when a DPoP proof needs to be signed, allowing the application to override the default signing implementation.
`OnDeviceCode`	Fires when the Device Code flow issues the user code that must be entered on a secondary device (RFC 8628).
`OnDeviceCodeExpired`	Fires when the device code expires before the user completes authorization.

`OnErrorAccessToken`	Fires when the token endpoint rejects the access-token request.
`OnErrorAuthorizeCode`	Fires when the authorization server returns an error during the authorization code step.
`OnErrorIntrospectToken`	property OnErrorIntrospectToken: TsgcOnAuth2ErrorIntrospectToken; // TsgcOnAuth2ErrorIntrospectToken = procedure(Sender: TObject; const Error, Error_Description, Error_URI, RawParams: String) of objec...
`OnErrorRefreshToken`	Fires when the token endpoint rejects the refresh_token grant.
`OnErrorRevokeToken`	property OnErrorRevokeToken: TsgcOnAuth2ErrorRevokeToken; // TsgcOnAuth2ErrorRevokeToken = procedure(Sender: TObject; const Error, Error_Description, Error_URI, RawParams: String) of object __property...
`OnHTTPResponse`	Fires just before the local HTTP server sends the browser-facing response after the authorization redirect.

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 (RFC 6749)** configuration sourced from the online help.

> **About this scenario.** The Authorization Code grant is the most common OAuth2 flow. It is designed for web applications, desktop applications, and mobile applications where the client can securely store a client secret. The flow involves redirecting the user to the authorization server, where they authenticate and grant permission. The server then returns an authorization code to a redirect URI, and the client exchanges that code for an access token.

Delphi (VCL / FireMonkey)

```
OAuth2.OAuth2Options.GrantType := auth2Code;
OAuth2.OAuth2Options.ClientId := 'your-client-id';
OAuth2.OAuth2Options.ClientSecret := 'your-client-secret';
OAuth2.AuthorizationServerOptions.AuthURL := 'https://provider.com/oauth2/authorize';
OAuth2.AuthorizationServerOptions.TokenURL := 'https://provider.com/oauth2/token';
OAuth2.AuthorizationServerOptions.Scope.Text := 'openid profile';
OAuth2.LocalServerOptions.IP := '127.0.0.1';
OAuth2.LocalServerOptions.Port := 8080;
OAuth2.Start;
```

C++ Builder

```
OAuth2->OAuth2Options->GrantType = auth2Code;
OAuth2->OAuth2Options->ClientId = "your-client-id";
OAuth2->OAuth2Options->ClientSecret = "your-client-secret";
OAuth2->AuthorizationServerOptions->AuthURL = "https://provider.com/oauth2/authorize";
OAuth2->AuthorizationServerOptions->TokenURL = "https://provider.com/oauth2/token";
OAuth2->AuthorizationServerOptions->Scope->Text = "openid profile";
OAuth2->LocalServerOptions->IP = "127.0.0.1";
OAuth2->LocalServerOptions->Port = 8080;
OAuth2->Start();
```

.NET (C#)

```
OAuth2.OAuth2Options.GrantType = TsgcOAuth2GrantType.auth2Code;
OAuth2.OAuth2Options.ClientId = "your-client-id";
OAuth2.OAuth2Options.ClientSecret = "your-client-secret";
OAuth2.AuthorizationServerOptions.AuthURL = "https://provider.com/oauth2/authorize";
OAuth2.AuthorizationServerOptions.TokenURL = "https://provider.com/oauth2/token";
OAuth2.AuthorizationServerOptions.Scope.Text = "openid profile";
OAuth2.LocalServerOptions.IP = "127.0.0.1";
OAuth2.LocalServerOptions.Port = 8080;
OAuth2.Start();
```

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 · Client Credentials Grant (RFC 6749)

The Client Credentials grant is used for machine-to-machine (M2M) authentication where no user interaction is required. The client authenticates directly with the authorization server using its own credentials (client_id and client_secret) and receives an access token. There is no authorization code step and no user login.

Delphi (VCL / FireMonkey)

```
OAuth2.OAuth2Options.GrantType := auth2ClientCredentials;
OAuth2.OAuth2Options.ClientId := 'your-client-id';
OAuth2.OAuth2Options.ClientSecret := 'your-client-secret';
OAuth2.AuthorizationServerOptions.TokenURL := 'https://provider.com/oauth2/token';
OAuth2.AuthorizationServerOptions.Scope.Text := 'api.read api.write';
OAuth2.Start;
```

C++ Builder

```
OAuth2->OAuth2Options->GrantType = auth2ClientCredentials;
OAuth2->OAuth2Options->ClientId = "your-client-id";
OAuth2->OAuth2Options->ClientSecret = "your-client-secret";
OAuth2->AuthorizationServerOptions->TokenURL = "https://provider.com/oauth2/token";
OAuth2->AuthorizationServerOptions->Scope->Text = "api.read api.write";
OAuth2->Start();
```

.NET (C#)

```
OAuth2.OAuth2Options.GrantType = TsgcOAuth2GrantType.auth2ClientCredentials;
OAuth2.OAuth2Options.ClientId = "your-client-id";
OAuth2.OAuth2Options.ClientSecret = "your-client-secret";
OAuth2.AuthorizationServerOptions.TokenURL = "https://provider.com/oauth2/token";
OAuth2.AuthorizationServerOptions.Scope.Text = "api.read api.write";
OAuth2.Start();
```

2 · Key Generation

You can generate an ES256 key pair directly in Delphi using the GenerateDPoPKeyPair method. This uses OpenSSL internally to create the key pair and automatically sets both PrivateKey and PublicKeyJWK:

Delphi (VCL / FireMonkey)

```
OAuth2.DPoPOptions.Enabled := True;
OAuth2.DPoPOptions.Algorithm := dpopES256;
OAuth2.GenerateDPoPKeyPair; // Generates PrivateKey + PublicKeyJWK automatically
OAuth2.Start;
```

C++ Builder

```
OAuth2->DPoPOptions->Enabled = true;
OAuth2->DPoPOptions->Algorithm = dpopES256;
OAuth2->GenerateDPoPKeyPair(); // Generates PrivateKey + PublicKeyJWK automatically
OAuth2->Start();
```

.NET (C#)

```
OAuth2.DPoPOptions.Enabled = true;
OAuth2.DPoPOptions.Algorithm = TsgcHTTPOAuth2_DPoP_Algorithm.dpopES256;
OAuth2.GenerateDPoPKeyPair(); // Generates PrivateKey + PublicKeyJWK automatically
OAuth2.Start();
```

3 · Resource Owner Password Credentials Grant (RFC 6749)

The Resource Owner Password Credentials (ROPC) grant allows the client to collect the user's username and password directly and exchange them for an access token. The user's credentials are sent to the token endpoint, and the server returns an access token if they are valid.

Delphi (VCL / FireMonkey)

```
OAuth2.OAuth2Options.GrantType := auth2ResourceOwnerPassword;
OAuth2.OAuth2Options.ClientId := 'your-client-id';
OAuth2.OAuth2Options.ClientSecret := 'your-client-secret';
OAuth2.OAuth2Options.UserName := 'user@example.com';
OAuth2.OAuth2Options.Password := 'user-password';
OAuth2.AuthorizationServerOptions.TokenURL := 'https://provider.com/oauth2/token';
OAuth2.AuthorizationServerOptions.Scope.Text := 'openid profile';
OAuth2.Start;
```

C++ Builder

```
OAuth2->OAuth2Options->GrantType = auth2ResourceOwnerPassword;
OAuth2->OAuth2Options->ClientId = "your-client-id";
OAuth2->OAuth2Options->ClientSecret = "your-client-secret";
OAuth2->OAuth2Options->UserName = "user@example.com";
OAuth2->OAuth2Options->Password = "user-password";
OAuth2->AuthorizationServerOptions->TokenURL = "https://provider.com/oauth2/token";
OAuth2->AuthorizationServerOptions->Scope->Text = "openid profile";
OAuth2->Start();
```

.NET (C#)

```
OAuth2.OAuth2Options.GrantType = TsgcOAuth2GrantType.auth2ResourceOwnerPassword;
OAuth2.OAuth2Options.ClientId = "your-client-id";
OAuth2.OAuth2Options.ClientSecret = "your-client-secret";
OAuth2.OAuth2Options.UserName = "user@example.com";
OAuth2.OAuth2Options.Password = "user-password";
OAuth2.AuthorizationServerOptions.TokenURL = "https://provider.com/oauth2/token";
OAuth2.AuthorizationServerOptions.Scope.Text = "openid profile";
OAuth2.Start();
```

4 · TsgcHTTP_OAuth2_Client_Google

This component lets you login with your Google Account in an easy way.

Delphi (VCL / FireMonkey)

```
procedure GoogleSignIn;
begin
var
  oClient: TsgcHTTP_OAuth2_Client_Google;
  oData: TsgcOAuth2_Google_Data;
begin
  oClient := TsgcHTTP_OAuth2_Client_Google.Create(nil);
  Try
    oData := oClient.Authenticate('client_id', 'client_secret');
    if oData.Authenticated then
      ShowMessage(oData.UserProfile._Name);
  Finally
    oClient.Free;
  End;
end;
```

C++ Builder

```
void GoogleSignIn()
{
  TsgcHTTP_OAuth2_Client_Google *oClient = new TsgcHTTP_OAuth2_Client_Google.Create(this);
  Try
  {
    TsgcOAuth2_Google_Data *oData = oClient.Authenticate("client_id", "client_secret");
    if oData→Authenticated() then
      ShowMessage(oData→UserProfile→_Name);
  }
  __Finally
    oClient→Free();
  }
}
```

.NET (C#)

```
void GoogleSignIn()
{
  TsgcHTTP_OAuth2_Client_Google oClient = new TsgcHTTP_OAuth2_Client_Google();
  TsgcOAuth2_Google_Data oData = oClient.Authenticate("client_id", "client_secret");
  if (oData.Authenticated)
    MessageBox.Show(oData.UserProfile._Name);
}
```

5 · TsgcHTTP_OAuth2_Client_Microsoft

This component lets you login with your Microsoft Account in an easy way.

Delphi (VCL / FireMonkey)

```
procedure MicrosoftSignIn;
begin
var
  oClient: TsgcHTTPComponentClient_OAuth2_Microsoft;
  oData: TsgcOAuth2_Microsoft_Data;
begin
  oClient := TsgcHTTPComponentClient_OAuth2_Microsoft.Create(nil);
  Try
    oData := oClient.Authenticate('tenant_id', 'client_id', 'client_secret');
    if oData.Authenticated then
      ShowMessage(oData.UserProfile.DisplayName);
  Finally
    oClient.Free;
  End;
end;
```

C++ Builder

```
void GoogleSignIn()
{
  TsgcHTTPComponentClient_OAuth2_Microsoft *oClient = new TsgcHTTPComponentClient_OAuth2_Microso
  Try
  {
    TsgcOAuth2_Microsoft_Data *oData = oClient.Authenticate("tenant_id", "client_id", "client_se
    if oData->Authenticated() then
      ShowMessage(oData->UserProfile->DisplayName);
  }
  __Finally
    oClient->Free();
  }
}
```

.NET (C#)

```
void MicrosoftSignIn()
{
  TsgcHTTPComponentClient_OAuth2_Microsoft oClient = new TsgcHTTPComponentClient_OAuth2_Microsof
  TsgcOAuth2_Microsoft_Data oData = oClient.Authenticate("tenant_id", "client_id", "client_secre
  if (oData.Authenticated)
    MessageBox.Show(oData.UserProfile.DisplayName);
}
```

6 · DPoP - Demonstrating Proof of Possession (RFC 9449)

DPoP (Demonstrating Proof of Possession) is a security mechanism that binds access tokens to a specific client by requiring proof of possession of a private key. Unlike bearer tokens, which can be used by anyone who obtains them, DPoP-bound tokens are useless without the corresponding private key. This provides protection against token theft and replay attacks.

Delphi (VCL / FireMonkey)

```
// Configure the grant type (DPoP works with any grant type)
OAuth2.OAuth2Options.GrantType := auth2CodePKCE;
OAuth2.OAuth2Options.ClientId := 'your-client-id';
OAuth2.AuthorizationServerOptions.AuthURL := 'https://provider.com/oauth2/authorize';
OAuth2.AuthorizationServerOptions.TokenURL := 'https://provider.com/oauth2/token';
OAuth2.AuthorizationServerOptions.Scope.Text := 'openid profile';
// Enable DPoP
OAuth2.DPoPOptions.Enabled := True;
OAuth2.DPoPOptions.Algorithm := 'ES256';
OAuth2.DPoPOptions.PrivateKey := LoadPrivateKeyFromFile('dpop_private.pem');
OAuth2.DPoPOptions.PublicKeyJWK := '{"kty":"EC","crv":"P-256","x":"...","y":"..."}';
OAuth2.Start;
// After obtaining the token, generate DPoP proofs for API calls
var vProof := OAuth2.GetDPoPProof('GET', 'https://api.provider.com/resource');
// Use the proof in your HTTP request:
// Authorization: DPoP <access_token>
// DPoP: <vProof>
```

C++ Builder

```
// Configure the grant type
OAuth2->OAuth2Options->GrantType = auth2CodePKCE;
OAuth2->OAuth2Options->ClientId = "your-client-id";
OAuth2->AuthorizationServerOptions->AuthURL = "https://provider.com/oauth2/authorize";
OAuth2->AuthorizationServerOptions->TokenURL = "https://provider.com/oauth2/token";
OAuth2->AuthorizationServerOptions->Scope->Text = "openid profile";
// Enable DPoP
OAuth2->DPoPOptions->Enabled = true;
OAuth2->DPoPOptions->Algorithm = "ES256";
OAuth2->DPoPOptions->PrivateKey = LoadPrivateKeyFromFile("dpop_private.pem");
OAuth2->DPoPOptions->PublicKeyJWK = "{\"kty\":\"EC\",\"crv\":\"P-256\",\"x\":\"...\",\"y\":\"...
OAuth2->Start();
// Generate DPoP proof for API calls
UnicodeString Proof = OAuth2->GetDPoPProof("GET", "https://api.provider.com/resource");
```

.NET (C#)

```
// Configure the grant type
OAuth2.OAuth2Options.GrantType = TsgcOAuth2GrantType.auth2CodePKCE;
OAuth2.OAuth2Options.ClientId = "your-client-id";
OAuth2.AuthorizationServerOptions.AuthURL = "https://provider.com/oauth2/authorize";
OAuth2.AuthorizationServerOptions.TokenURL = "https://provider.com/oauth2/token";
OAuth2.AuthorizationServerOptions.Scope.Text = "openid profile";
// Enable DPoP
OAuth2.DPoPOptions.Enabled = true;
OAuth2.DPoPOptions.Algorithm = "ES256";
OAuth2.DPoPOptions.PrivateKey = File.ReadAllText("dpop_private.pem");
OAuth2.DPoPOptions.PublicKeyJWK = "{\"kty\":\"EC\",\"crv\":\"P-256\",\"x\":\"...\",\"y\":\"...\"
OAuth2.Start();
// Generate DPoP proof for API calls
string proof = OAuth2.GetDPoPProof("GET", "https://api.provider.com/resource");
// Use: Authorization: DPoP <access_token>
// Use: DPoP: <proof>
```

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 — PKCE — RFC 7636	datatracker.ietf.org/doc/html/rfc7636
Online help — component page	www.esegece.com/help/sgcWebSockets/Components/HTTP/Authorization/OAuth2/client/TsgcHTTP_OAuth2_Client.htm
Delphi demo project (in the sgcWebSockets package)	`Demos\20.HTTP_Protocol\02.OAuth2_Authentication`
Component page	www.esegece.com/products/websockets/http/oauth2-client/
Product page	www.esegece.com/products/websockets/

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