SGCWEBSOCKETS · TECHNICAL DOCUMENT

# OpenID Connect Client

TsgcHTTP_OIDC_Client: OpenID Connect sign in with discovery, PKCE, nonce and strict ID token validation.

eSeGeCe

WWW.ESEGECE.COM

## Overview

Sign users in with Google, Microsoft Entra ID, Okta, Auth0, Keycloak or AWS Cognito, and know who they are. The component discovers the provider, runs the browser flow with PKCE and nonce, and validates the ID token it gets back.

**TsgcHTTP_OIDC_Client** signs users in with any **OpenID Connect** provider (Google, Microsoft Entra ID, Okta, Auth0, Keycloak, AWS Cognito and others) and validates the **ID token** they return. It extends the TsgcHTTP_OAuth2_Client: the authorization code flow, the loopback server, the token requests, refresh and DPoP work as in OAuth 2.0, and the component adds discovery, PKCE and nonce by default, validation of the ID token against the keys of the provider, and the userinfo endpoint.

The unit also contains TsgcOIDCJWKS, a thread safe cache of the signing keys of an issuer, and the function sgcOIDC_ValidateIDToken, which validates tokens on the server side, for example the bearer tokens received by a REST API.

## At a glance

| | |
|---|---|
| COMPONENT CLASS<br>`TsgcHTTP_OIDC_Client` | STANDARDS / SPEC<br>**OpenID Connect Core 1.0 · OAuth 2.0 with PKCE** |
| TRANSPORTS<br>**HTTPS (OpenSSL or SChannel)** | PLATFORMS<br>**Windows, macOS, Linux, iOS, Android** |
| FRAMEWORKS<br>**VCL, FireMonkey** | EDITION<br>**Enterprise (also sgcAuth pack)** |

## Features

- **Discovery.** `Discover` reads the provider configuration of `OIDCOptions.Issuer` and fills the authorization and token URLs, `JWKSURI`, `UserInfoEndpoint` and `EndSessionEndpoint`. The raw JSON stays in `DiscoveryDocument`.

- **PKCE and nonce by default.** `OIDCOptions.UsePKCE` is on out of the box and every sign in sends a fresh `Nonce` that the ID token must echo, so an intercepted code or a replayed token is useless.
- **Strict ID token validation.** Signature, issuer, audience, expiry (with `ClockSkew`) and nonce are checked. Only RS256, RS384, RS512, ES256 and ES384 are accepted, `none` and the HS algorithms are always rejected. Results in `IDToken`, `IDTokenClaims` and `IDTokenValid`.
- **Key rotation handled.** `TsgcOIDCJWKS` is a thread safe cache of the provider signing keys. A token with an unknown key id triggers a new download, limited by `RefetchInterval`, so key rotation needs no restart.
- **Server-side validation.** `sgcOIDC_ValidateIDToken` validates the bearer tokens your REST API or WebSocket server receives, against the same JWKS cache. `OIDCOptions.AllowedTenants` restricts multi-tenant Entra ID apps to the organizations you accept.
- **Userinfo and the OAuth2 toolbox.** `GetUserInfo` returns the profile JSON. Loopback redirect server, refresh tokens, DPoP, device code, revocation and introspection come from `TsgcHTTP_OAuth2_Client`.

# Technical specification

| | |
|---|---|
| **Standards & specs** | OpenID Connect Core 1.0 · OpenID Connect Discovery 1.0 · RFC 7636 · RFC 7517 · RFC 7519 |
| **Component class** | `TsgcHTTP_OIDC_Client` (unit `sgcAuth_OIDC_Client`) |
| **Frameworks** | VCL, FireMonkey |
| **Platforms** | Windows, macOS, Linux, iOS, Android |
| **Edition** | sgcWebSockets Enterprise, also sold in the sgcAuth pack |

## Main properties

The published and public properties used to configure and drive the component.

| | |
|---|---|
| `DiscoveryDocument` | JSON of the last discovery document read by Discover. |
| `EndSessionEndpoint` | URL of the end session (logout) endpoint, from the discovery document. |
| `IDToken` | The raw ID token of the last token response. |
| `IDTokenClaims` | JSON claims of the last validated ID token. |
| `IDTokenValid` | True when the last ID token passed the validation. |
| `JWKS` | Cache of the signing keys of the provider used to validate the ID tokens. |
| `JWKSURI` | URL of the JSON Web Key Set of the provider, from the discovery document. |
| `Nonce` | Nonce sent in the authorization request of the current sign in. |
| `OIDCOptions` | OpenID Connect settings: issuer, PKCE, ID token validation, clock skew and allowed tenants. |
| `UserInfoEndpoint` | URL of the userinfo endpoint, from the discovery document. |

## Main methods

The public methods exposed by the component.

| | |
|---|---|
| `Discover` | Reads the OpenID Connect discovery document of the issuer and configures the endpoints. |
| `GetUserInfo` | Calls the userinfo endpoint with an access token and returns the JSON response. |
| `ValidateIDToken` | Validates an ID token with the configuration of the component. |

## Events

The events fired by the component.

| | |
|---|---|
| `OnOIDCIDToken` | Fires after the ID token of a token response has been validated. |

# Quick Start

Set OIDCOptions.Issuer and the client credentials, call Start, and read the validated claims in OnOIDCIDToken. Discovery runs automatically when the endpoints are not set.

> **About this scenario.** Set the issuer, start the sign in. The same code ships in the demo `Demos\26.Authentication\04.OpenID_Connect`.

### Delphi (VCL / FireMonkey)

```
uses
  sgcAuth_OIDC_Client;

// OIDC is a form field: OIDC: TsgcHTTP_OIDC_Client;
procedure TForm1.SignIn;
begin
  OIDC := TsgcHTTP_OIDC_Client.Create(nil);
  OIDC.OnOIDCIDToken := OnOIDCIDTokenEvent;

  OIDC.OIDCOptions.Issuer := 'https://accounts.google.com';
  OIDC.OAuth2Options.ClientId := 'your-client-id';
  OIDC.OAuth2Options.ClientSecret := 'your-client-secret';
  OIDC.AuthorizationServerOptions.Scope.Clear;
  OIDC.AuthorizationServerOptions.Scope.Add('openid');
  OIDC.AuthorizationServerOptions.Scope.Add('profile');
  OIDC.AuthorizationServerOptions.Scope.Add('email');

  OIDC.LocalServerOptions.IP := '127.0.0.1';
  OIDC.LocalServerOptions.Port := 8080;
  OIDC.LocalServerOptions.RedirectURL := 'http://127.0.0.1:8080/';

  // discovery, then the browser sign in with PKCE and nonce
  OIDC.Start;
end;

procedure TForm1.OnOIDCIDTokenEvent(Sender: TObject; const aClaims: string;
  aValid: Boolean; const aError: string);
begin
  if aValid then
    Memo1.Lines.Text := aClaims // JSON with sub, email, name...
  else
    ShowMessage(aError);
end;
```

## C++ Builder

```
// uses: sgcAuth_OIDC_Client
TsgcHTTP_OIDC_Client *OIDC = new TsgcHTTP_OIDC_Client(this);
OIDC->OnOIDCIDToken = OnOIDCIDTokenEvent;

OIDC->OIDCOptions->Issuer = "https://accounts.google.com";
OIDC->OAuth2Options->ClientId = "your-client-id";
OIDC->OAuth2Options->ClientSecret = "your-client-secret";
OIDC->AuthorizationServerOptions->Scope->Clear();
OIDC->AuthorizationServerOptions->Scope->Add("openid");
OIDC->AuthorizationServerOptions->Scope->Add("profile");
OIDC->AuthorizationServerOptions->Scope->Add("email");

OIDC->LocalServerOptions->IP = "127.0.0.1";
OIDC->LocalServerOptions->Port = 8080;
OIDC->LocalServerOptions->RedirectURL = "http://127.0.0.1:8080/";

OIDC->Start();
```

## Common scenarios

The following topics come from the online help. Each one explains a part of the component and shows the Delphi code that drives it.

### 1 · TLS and certificate verification

The discovery, JWKS, userinfo and token requests verify the server certificate by default (**HTTPClientOptions.TLSOptions.VerifyCertificate** is `True`). With the OpenSSL IO handler, OpenSSL does not read the Windows certificate store, so set **HTTPClientOptions.TLSOptions.RootCertFile** to a PEM file with the trusted root CA certificates (for example a `cacert.pem` bundle from curl.se), or on Windows set **HTTPClientOptions.TLSOptions.IOHandler := iohSChannel** to use the Windows certificate store instead. Without one of the two, the requests fail with an error that names the failing URL and explains that OpenSSL has no trusted root CA certificates.

### 2 · Discovery

Set **OIDCOptions.Issuer** and call **Discover**. The component reads `<Issuer>/.well-known/openid-configuration` over https, checks that the `issuer` of the document is the configured issuer, and sets **AuthorizationServerOptions.AuthURL**, **AuthorizationServerOptions.TokenURL**, **JWKSURI**, **UserInfoEndpoint** and **EndSessionEndpoint**. You do not need to call it before a sign in: **Start** discovers the provider when the issuer has not been discovered yet.

Typical issuers: `https://accounts.google.com`, `https://login.microsoftonline.com/<tenant-id>/v2.0`, `https://<domain>.okta.com`, `https://<tenant>.auth0.com/` and `https://<host>/realms/<realm>` for Keycloak. Type the issuer exactly as the provider publishes it, including a final slash when it has one.

### 3 · Sign in with PKCE and nonce

Register the application with the provider as a desktop or native application with a loopback redirect URL, then configure the client id, the scopes and the loopback server and call **Start**:

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

```
oOIDC := TsgcHTTP_OIDC_Client.Create(nil);
oOIDC.OIDCOptions.Issuer := 'https://accounts.google.com';
oOIDC.OAuth2Options.ClientId := 'your-client-id';
oOIDC.OAuth2Options.ClientSecret := 'your-client-secret';
oOIDC.AuthorizationServerOptions.Scope.Add('email');
oOIDC.AuthorizationServerOptions.Scope.Add('profile');
// ... loopback server that receives the authorization code
oOIDC.LocalServerOptions.IP := '127.0.0.1';
oOIDC.LocalServerOptions.Port := 8080;
oOIDC.LocalServerOptions.RedirectURL := 'http://127.0.0.1:8080';
// ... OpenSSL 3 for the discovery, token, JWKS and userinfo requests
oOIDC.HTTPClientOptions.TLSOptions.OpenSSL_Options.APIVersion := oslAPI_3_0;
// ... trusted root CA certificates, verification is on by default
oOIDC.HTTPClientOptions.TLSOptions.RootCertFile := 'cacert.pem';
oOIDC.OnOIDCIDToken := OnOIDCIDTokenEvent;
// ... discovers the endpoints, opens the browser and waits for the code
oOIDC.Start;
```

1. **Start** discovers the provider and adds the `openid` scope when it is missing.
2. With **OIDCOptions.UsePKCE** (the default) the authorization code grant becomes authorization code with PKCE ( `auth2CodePKCE` ).
3. A random `nonce` is created for every sign in and added to the authorization URL, then the browser opens the sign in page of the provider.
4. The provider redirects to the loopback server with the code and the component exchanges it for the access token and the ID token.
5. With **OIDCOptions.ValidateIDToken** (the default) the ID token is validated, including the nonce, and **OnOIDCIDToken** fires. **IDToken**, **IDTokenClaims** and **IDTokenValid** keep the result.

## 4 · OnOIDCIDToken

The event receives the claims of the ID token as a JSON string, whether the token is valid and the reason when it is not. Sign the user in only when `aValid` is True, and identify the user by the `sub` claim (together with the issuer), not by the email address.

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

```
procedure TForm1.OnOIDCIDTokenEvent(Sender: TObject; const aClaims: string;
  aValid: Boolean; const aError: string);
var
  oJSON: TsgcJSON;
begin
  if not aValid then
  begin
    WriteToLog('ID token rejected: ' + aError);
    Exit;
  end;
  // ... signature, iss, aud, azp, exp, iat and nonce already checked
  oJSON := TsgcJSON.Create(nil);
  try
    oJSON.Read(aClaims);
    SignInUser(oJSON.Node['sub'].Value, oJSON.Node['email'].Value);
  finally
    oJSON.Free;
  end;
end;
```

The event runs in the thread of the loopback server, synchronize with the main thread before you update the user interface.

### 5 · UserInfo

After the sign in, **GetUserInfo** calls the userinfo endpoint of the provider with the access token and returns its JSON response, for example the name, the picture and the email of the user. Pass an access token to use a different one.

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

```
Memo1.Lines.Text := oOIDC.GetUserInfo;
```

### 6 · Microsoft Entra ID multi-tenant applications

The issuers `https://login.microsoftonline.com/common/v2.0`, `.../organizations/v2.0` and `.../consumers/v2.0` do not identify one organization: their discovery document publishes the issuer as the template `https://login.microsoftonline.com/{tenantid}/v2.0`. The component accepts that template, and when it validates an ID token it requires a `tid` claim with a GUID value and an `iss` claim equal to the template filled with that `tid`.

Such a configuration accepts users of **any** Entra ID organization. Add the tenant ids of the organizations you accept to **OIDCOptions.AllowedTenants**: when the list is not empty, a token whose `tid` is not in the list is rejected with "Tenant ... is not an allowed tenant.". Personal Microsoft accounts use the tenant `9188040d-6c67-4c5b-b112-36a304b66dad`.

Delphi (VCL / FireMonkey)

```
// ... any Entra ID organization: the issuer is a {tenantid} template
oOIDC.OIDCOptions.Issuer := 'https://login.microsoftonline.com/organizations/v2.0';
oOIDC.OAuth2Options.ClientId := '11111111-2222-3333-4444-555555555555';
// ... accept only the tenants of your customers
oOIDC.OIDCOptions.AllowedTenants.Add('aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee');
oOIDC.OIDCOptions.AllowedTenants.Add('ffffffff-0000-1111-2222-333333333333');
oOIDC.Start;
```

## 7 · Server-side validation: sgcOIDC_ValidateIDToken

A REST API or a WebSocket server that receives a token in the `Authorization: Bearer` header must check its signature and claims before it trusts it. The function **sgcOIDC_ValidateIDToken** does it with the keys of a TsgcOIDCJWKS cache. It is declared in the unit `sgcAuth_OIDC_Client`, and it is the same validation used by the component for the ID tokens.

Delphi (VCL / FireMonkey)

```
function sgcOIDC_ValidateIDToken(const aToken: string;
  const aJWKS: TsgcOIDCJWKS; const aIssuer, aClientId, aNonce: string;
  out aClaims: string; out aError: string; aClockSkew: Integer = 120;
  const aAllowedTenants: TStrings = nil): Boolean;
```

| Name | Type | Description |
|---|---|---|
| `aToken` | `const string` | The token in compact JWS format (header.payload.signature), at most 65536 characters. |
| `aJWKS` | `TsgcOIDCJWKS` | Key cache of the issuer. When its **JWKSURL** is set, the keys are downloaded on first use and when an unknown `kid` appears. |
| `aIssuer` | `const string` | Expected `iss` claim. It can contain `{tenantid}` for Entra ID multi-tenant issuers. |
| `aClientId` | `const string` | Expected audience: the `aud` claim must contain it. |
| `aNonce` | `const string` | Expected `nonce` claim. Pass an empty string when the token has no nonce to check, for example an access token. |
| `aClaims` | `out string` | The JSON payload of the token when it is valid, empty otherwise. |

| | | |
|---|---|---|
| `aError` | `out string` | Reason of the failure, empty when the token is valid. |
| `aClockSkew` | `Integer` | Tolerance in seconds for `exp`, `nbf` and `iat`. Default 120. |
| `aAllowedTenants` | `TStrings` | Optional list of accepted `tid` values. Nil or empty accepts any tenant. |

The function returns True when the token is valid. It never raises an exception and it can be called from several threads with the same **TsgcOIDCJWKS**. Validate only tokens issued for your application: ID tokens, or access tokens that the provider issues for your own API (the audience is then the identifier of the API).

Delphi (VCL / FireMonkey)

```
// ... once, at startup: discovery gives the JWKS URL of the issuer
FOIDC := TsgcHTTP_OIDC_Client.Create(nil);
FOIDC.OIDCOptions.Issuer := 'https://accounts.google.com';
FOIDC.HTTPClientOptions.TLSOptions.OpenSSL_Options.APIVersion := oslAPI_3_0;
FOIDC.HTTPClientOptions.TLSOptions.RootCertFile := 'cacert.pem';
FOIDC.Discover;
FOIDC.JWKS.TLSOptions := FOIDC.HTTPClientOptions.TLSOptions;

// ... in the request handler of your API (any thread)
procedure TMyAPI.OnCommandGet(AContext: TIdContext;
  ARequestInfo: TIdHTTPRequestInfo; AResponseInfo: TIdHTTPResponseInfo);
var
  vToken, vClaims, vError: string;
begin
  vToken := ARequestInfo.RawHeaders.Values['Authorization'];
  if not SameText(Copy(vToken, 1, 7), 'Bearer ') then
  begin
    AResponseInfo.ResponseNo := 401;
    Exit;
  end;
  vToken := Trim(Copy(vToken, 8, Length(vToken)));
  // ... signature (JWKS), iss, aud, exp, nbf and iat
  if sgcOIDC_ValidateIDToken(vToken, FOIDC.JWKS, 'https://accounts.google.com',
    'your-client-id', '', vClaims, vError) then
  begin
    AResponseInfo.ContentType := 'application/json';
    AResponseInfo.ContentText := ProcessRequest(ARequestInfo, vClaims);
  end
  else
  begin
    AResponseInfo.ResponseNo := 401;
    WriteToLog('token rejected: ' + vError);
  end;
end;
```

## 8 · Accepted algorithms and claim checks

- **Format.** The token must be a signed compact JWS with three Base64url parts. Headers with a `crit` parameter are rejected.
- **Algorithms.** Only `RS256`, `RS384`, `RS512`, `ES256` and `ES384` are accepted. `none` and the symmetric `HS256`, `HS384` and `HS512` are always rejected, so a token can not be forged with the public key used as an HMAC secret.
- **Key selection.** The key is chosen by the `kid` of the header and its type, algorithm and curve must match the algorithm of the token. A token without `kid` is accepted only when exactly one key matches. Keys marked for encryption (`use` other than `sig`), RSA keys shorter than 2048 bits, symmetric keys and unsupported curves are ignored.
- **Key rotation.** An unknown `kid` downloads the JWKS again, at most once every **RefetchInterval** seconds (60 by default).
- **iss.** Must be equal to the expected issuer, or match the `{tenantid}` template with the GUID of the `tid` claim.
- **tid.** Must be in the allowed tenants when the list is not empty.
- **aud and azp.** The audience must contain the client id. When it contains several values, `azp` is required and must be the client id.
- **exp.** Required, the token is rejected after `exp` plus the clock skew.
- **nbf.** Optional, the token is rejected before `nbf` minus the clock skew.
- **iat.** Required, it can not be in the future beyond the clock skew.
- **nonce.** When an expected nonce is given, the claim must be present and equal (constant time comparison).
- **Secure URLs.** The discovery document, the JWKS and the userinfo endpoint are only downloaded over https (http is only allowed for `localhost`, `127.0.0.1` and `[::1]`) and the size of the documents is limited.

## 9 · Inherited members

The OAuth 2.0 members are documented in the TsgcHTTP_OAuth2_Client reference: the properties AuthorizationServerOptions, OAuth2Options, LocalServerOptions, HTTPClientOptions and DPoPOptions, the methods Start, Stop, Refresh, Revoke and Introspect, and the events OnBeforeAuthorizeCode, OnAfterAccessToken, OnErrorAccessToken and the others. A refreshed ID token is validated too, without nonce, when the token response contains one.

## Sources used to build this document

Every external claim links back to a primary source. The online help references are the canonical pages the company maintains for this component.

| | |
|---|---|
| **Primary standard / spec: OpenID Connect Core 1.0** | openid.net/specs/openid-connect-core-1_0.html |
| **Primary standard / spec: OpenID Connect Discovery 1.0** | openid.net/specs/openid-connect-discovery-1_0.html |
| **Primary standard / spec: RFC 7636** | datatracker.ietf.org/doc/html/rfc7636 |
| **Primary standard / spec: RFC 7517** | datatracker.ietf.org/doc/html/rfc7517 |
| **Primary standard / spec: RFC 7519** | datatracker.ietf.org/doc/html/rfc7519 |
| **Online help: component page** | www.esegece.com/help/sgcWebSockets/ (TsgcHTTP_OIDC_Client) |
| **Delphi demo project (in the sgcWebSockets package)** | `Demos\26.Authentication\04.OpenID_Connect` |
| **Component page** | www.esegece.com/products/websockets/http/oidc-client/ |
| **sgcAuth pack** | www.esegece.com/products/sgcauth/ |
| **Product page** | www.esegece.com/products/websockets/ |

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