SGCWEBSOCKETS · TECHNICAL DOCUMENT

Mail OAuth2

TsgcMailOAuth2: OAuth2 tokens and SASL XOAUTH2 / OAUTHBEARER strings for SMTP, IMAP and POP3 with Microsoft 365 and Gmail.

eSeGeCe

WWW.ESEGECE.COM

Overview

Microsoft 365 and Gmail no longer accept a plain password for most mailboxes. Get the OAuth2 tokens, keep them fresh, and hand your SMTP, IMAP or POP3 client the exact SASL string it needs.

The **TsgcMailOAuth2** component obtains and refreshes the OAuth 2.0 access tokens needed to send and read mail with **Microsoft 365** (Exchange Online) and **Gmail**, and builds the SASL **XOAUTH2** and **OAUTHBEARER** (RFC 7628) strings used by SMTP, IMAP and POP3. Both providers have disabled basic authentication with a password for most accounts, so a mail client needs OAuth2 to sign in.

The component is transport agnostic, it never talks to the mail server. Use it with Indy `TIdSMTP`, `TIdIMAP4`, `TIdPOP3` or any other mail library which can send a raw SASL command.

At a glance

COMPONENT CLASS
`TsgcMailOAuth2`

STANDARDS / SPEC
SASL OAUTHBEARER, RFC 7628 · XOAUTH2

TRANSPORTS
HTTPS for the token requests; SMTP, IMAP and POP3 through your mail library

PLATFORMS
Windows, macOS, Linux, iOS, Android

FRAMEWORKS
VCL, FireMonkey

EDITION
Enterprise (also sgcAuth pack)

Features

- **Microsoft 365 and Gmail presets.** `Provider` selects `mopMicrosoft365` or `mopGmail` with the right endpoints. `mopCustom` takes `CustomAuthURL`, `CustomTokenURL`, `CustomDeviceAuthorizationURL` and `CustomScope` for any other provider.
- **Scopes from the protocols.** Set `Protocols` to any mix of `mpSMTP`, `mpIMAP` and `mpPOP3` and the component requests the matching scopes, with `offline_access` on Microsoft 365. `GetScope` shows what will be asked.

- **Browser or device code.** `Flow` chooses `mofAuthorizationCodePKCE` (browser plus loopback redirect, see `LocalServerOptions`) or `mofDeviceCode` for services and consoles, where `OnDeviceCode` hands you the code and URL to show.
- **Token lifecycle.** `AccessToken`, `RefreshToken` and `ExpiresAt` are always current. `Refresh` renews the access token synchronously and `OnTokensChanged` fires every time, so you can persist the new refresh token.
- **SASL strings ready to send.** `GetXOAuth2` and `GetOAuthBearer` return the Base64 initial responses for `AUTH XOAUTH2` and `AUTH OAUTHBEARER`. The `Raw` variants and the `sgcGetXOAuth2` functions help with debugging and other token sources.
- **Transport agnostic.** The component never opens a mail connection. Use Indy `TIdSMTP`, `TIdIMAP4` or `TIdPOP3`, or any mail library that can send a raw SASL command, with `HTTPClientOptions` for the TLS of the token requests.

Technical specification

Standards & specs	RFC 7628 · XOAUTH2 SASL mechanism · OAuth for IMAP, POP and SMTP · RFC 7636 · RFC 8628
Component class	`TsgcMailOAuth2` (unit `sgcAuth_Mail_OAuth2`)
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.

`AccessToken`	Current access token, read-only.
`ClientId`	Client ID of the application registered with the provider.
`ClientSecret`	Client secret of the application, when the provider issued one.
`CustomAuthURL`	Authorization endpoint used when Provider is mopCustom.
`CustomDeviceAuthorizationURL`	Device authorization endpoint used when Provider is mopCustom and Flow is mofDeviceCode.
`CustomScope`	Space separated scopes requested when Provider is mopCustom.
`CustomTokenURL`	Token endpoint used when Provider is mopCustom.
`ExpiresAt`	UTC date and time when the access token expires, 0 when unknown.
`Flow`	OAuth2 flow used by Start: authorization code with PKCE or device code.
`HTTPClientOptions`	Options of the internal HTTP client which calls the token endpoint.
`LocalServerOptions`	Local HTTP server which receives the redirect of the authorization code flow.

`Protocols`	Mail protocols the access token is requested for (Microsoft 365).
`Provider`	Mail provider: Microsoft 365, Gmail or a custom OAuth2 server.
`RefreshToken`	Refresh token, assign the stored value before calling Refresh.
`TenantId`	Microsoft Entra tenant used in the Microsoft 365 endpoints.
`Version`	Read-only string exposing the sgcWebSockets library version.

Main methods

The public methods exposed by the component.

`GetAuthURL`	Returns the authorization endpoint for the current Provider.
`GetDeviceAuthorizationURL`	Returns the device authorization endpoint for the current Provider.
`GetOAuthBearer`	Returns the Base64 SASL OAUTHBEARER (RFC 7628) initial response.
`GetOAuthBearerRaw`	Returns the SASL OAUTHBEARER initial response without Base64 encoding.
`GetScope`	Returns the scope requested for the current Provider and Protocols.
`GetTokenURL`	Returns the token endpoint for the current Provider.
`GetXOAuth2`	Returns the Base64 SASL XOAUTH2 initial response for a user and the current access token.
`GetXOAuth2Raw`	Returns the SASL XOAUTH2 initial response without Base64 encoding.
`Refresh`	Requests a new access token with the RefreshToken, synchronously.
`Start`	Starts the configured flow: opens the browser (PKCE) or requests a device code.
`Stop`	Stops a running flow and the local HTTP server.

Events

The events fired by the component.

`OnDeviceCode`	Fired by the device code flow with the code the user must enter on the verification page.

`OnError`	Fired when the provider returns an OAuth2 error or the device code expires.
`OnTokensChanged`	Fired when new tokens are received, after Start and after every Refresh.

Quick Start

Pick the Provider and the Protocols, set ClientId, reuse the stored refresh token or call Start, then authenticate the mail session with GetXOAuth2.

> **About this scenario.** Sign in once, send mail for months. The same code ships in the demo `Demos\26.Authentication\06.Mail_OAuth2`.

Delphi (VCL / FireMonkey)

```
uses
  IdSMTP, sgcAuth_Mail_OAuth2;

// Mail is a form field: Mail: TsgcMailOAuth2;
procedure TForm1.FormCreate(Sender: TObject);
begin
  Mail := TsgcMailOAuth2.Create(Self);
  Mail.Provider := mopMicrosoft365;
  Mail.Protocols := [mpSMTP];
  Mail.TenantId := 'contoso.onmicrosoft.com';
  Mail.ClientId := 'your-application-id';
  Mail.OnTokensChanged := OnMailTokensChanged;

  // reuse the stored refresh token, sign in only when it no longer works
  Mail.RefreshToken := LoadRefreshToken;
  if (Mail.RefreshToken = '') or not Mail.Refresh then
    Mail.Start; // opens the browser, PKCE and a loopback redirect
end;

procedure TForm1.OnMailTokensChanged(Sender: TObject; const AccessToken,
  RefreshToken: String; const ExpiresAt: TDateTime);
begin
  SaveRefreshToken(RefreshToken); // encrypt it in your store
end;

// TIdSMTP with UseTLS = utUseExplicitTLS and AuthType = satNone
procedure TForm1.Authenticate(aSMTP: TIdSMTP);
begin
  aSMTP.Connect; // EHLO and STARTTLS
  aSMTP.SendCmd('AUTH XOAUTH2 ' + Mail.GetXOAuth2('user@contoso.com'), 235);
end;
```

C++ Builder

```
// uses: IdSMTP, sgcAuth_Mail_OAuth2
TsgcMailOAuth2 *Mail = new TsgcMailOAuth2(this);
Mail->Provider = mopMicrosoft365;
Mail->Protocols = TsgcMailOAuth2Protocols() << mpSMTP;
Mail->TenantId = "contoso.onmicrosoft.com";
Mail->ClientId = "your-application-id";
Mail->OnTokensChanged = OnMailTokensChanged;

Mail->RefreshToken = LoadRefreshToken();
if (Mail->RefreshToken.IsEmpty() || !Mail->Refresh())
  Mail->Start();

IdSMTP1->Connect();
IdSMTP1->SendCmd("AUTH XOAUTH2 " + Mail->GetXOAuth2("user@contoso.com"), 235);
```

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 · Microsoft 365 setup

1. In the Azure portal open **Microsoft Entra ID** › **App registrations** › **New registration** and register the application.
2. In **Authentication** add the platform **Mobile and desktop applications** with the loopback redirect URI of **LocalServerOptions** (for example `http://localhost`). For the device code flow enable **Allow public client flows**.
3. In **API permissions** add the delegated permissions for the protocols you use: `SMTP.Send`, `IMAP.AccessAsUser.All`, `POP.AccessAsUser.All`, plus `offline_access`.
4. Copy the **Application (client) ID** to **ClientId**. For a single tenant application copy the **Directory (tenant) ID** to **TenantId**, otherwise keep `common`.
5. SMTP AUTH must be enabled for the mailbox, for example with the Exchange Online PowerShell command `Set-CASMailbox -Identity user@contoso.com -SmtpClientAuthenticationDisabled $false`.

Servers: SMTP `smtp.office365.com:587` (STARTTLS), IMAP `outlook.office365.com:993`, POP3 `outlook.office365.com:995`.

2 · Gmail setup

1. In the Google Cloud Console create a project and configure the **OAuth consent screen** with the scope `https://mail.google.com/`. While the application is in testing mode, add the accounts which will sign in as test users.
2. In **Credentials** create an **OAuth client ID** of type **Desktop app**.
3. Copy the client ID to **ClientId** and the client secret to **ClientSecret**.
4. Use the authorization code flow with PKCE. Google does not allow the mail scope with the device code flow, **Start** raises an exception for that combination.

Servers: SMTP `smtp.gmail.com:587` (STARTTLS), IMAP `imap.gmail.com:993`, POP3 `pop.gmail.com:995`.

3 · Flows

- `mofAuthorizationCodePKCE` (default): **Start** opens the browser on the sign in page of the provider and a local HTTP server, configured in **LocalServerOptions**, receives the redirect. The tokens are then requested and **OnTokensChanged** is fired.
- `mofDeviceCode` : for applications without a browser, services or consoles. **Start** fires **OnDeviceCode** with a short user code and a verification URI. Show them to the user, who signs in on any other device. When the sign in is complete **OnTokensChanged** is fired, and if the code expires first **OnError** is fired with `expired_token` . Microsoft 365 only.

With `mopCustom` the endpoints and the scope come from **CustomAuthURL**, **CustomTokenURL**, **CustomDeviceAuthorizationURL** and **CustomScope**.

4 · Token persistence and OnTokensChanged

The user signs in only once. **OnTokensChanged** is fired every time new tokens are received, after **Start** and after every **Refresh**, with the access token, the refresh token and the UTC expiry time. Save the **RefreshToken** encrypted: it gives access to the mailbox. In the next session assign it to the **RefreshToken** property and call **Refresh**, which is synchronous, to get a new access token without user interaction. Access tokens are short lived (about one hour), refresh before **ExpiresAt**.

The token events can be fired in a secondary thread (the thread of the local HTTP server), synchronize any access to the user interface.

5 · SASL XOAUTH2 and OAUTHBEARER

GetXOAuth2 returns the Base64 initial client response of the XOAUTH2 mechanism, `user={user}^Aauth=Bearer {token}^A^A` , supported by Microsoft 365 and Gmail. **GetOAuthBearer** returns the standard OAUTHBEARER response of RFC 7628, `n,a={user},^Ahost={host}^Aport={port}^Aauth=Bearer {token}^A^A` , supported by Gmail. The **Raw** variants return the same strings without Base64 encoding, for libraries which encode them. Send them as:

- SMTP: `AUTH XOAUTH2 {GetXOAuth2}` , success reply 235.
- IMAP: `AUTHENTICATE XOAUTH2 {GetXOAuth2}` or `AUTHENTICATE OAUTHBEARER {GetOAuthBearer}` .
- POP3: `AUTH XOAUTH2` , then the string when the server answers `+` .

The functions `sgcGetXOAuth2` , `sgcGetXOAuth2Raw` , `sgcGetOAuthBearer` and `sgcGetOAuthBearerRaw` of the same unit build the strings from an access token obtained in any other way.

6 · Example: sending mail with Indy TIdSMTP

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

```
uses
  IdSMTP, IdMessage, IdSSLOpenSSL, IdExplicitTLSClientServerBase, sgcAuth_Mail_OAuth2;

procedure TForm1.FormCreate(Sender: TObject);
begin
  oMail := TsgcMailOAuth2.Create(Self);
  oMail.Provider := mopMicrosoft365;
  oMail.Protocols := [mpSMTP];
  oMail.TenantId := 'contoso.onmicrosoft.com';
  oMail.ClientId := '00000000-0000-0000-0000-000000000000';
  oMail.OnTokensChanged := OnMailTokensChanged;
  oMail.OnError := OnMailError;
  // next sessions: reuse the stored refresh token
  oMail.RefreshToken := LoadRefreshToken; // decrypted from your store
  if (oMail.RefreshToken = '') or not oMail.Refresh then
    oMail.Start; // opens the browser, the tokens arrive in OnTokensChanged
end;

procedure TForm1.OnMailTokensChanged(Sender: TObject; const AccessToken,
  RefreshToken: String; const ExpiresAt: TDateTime);
begin
  SaveRefreshToken(RefreshToken); // encrypt it
end;

procedure TForm1.OnMailError(Sender: TObject; const Error, ErrorDescription: String);
begin
  Log('OAuth2 error: ' + Error + ' ' + ErrorDescription);
end;

procedure TForm1.SendMail;
var
  oSMTP: TIdSMTP;
  oSSL: TIdSSLIOHandlerSocketOpenSSL;
  oMessage: TIdMessage;
begin
  oSMTP := TIdSMTP.Create(nil);
  oMessage := TIdMessage.Create(nil);
  try
    oSSL := TIdSSLIOHandlerSocketOpenSSL.Create(oSMTP);
    oSMTP.IOHandler := oSSL;
    oSMTP.Host := 'smtp.office365.com';
    oSMTP.Port := 587;
    oSMTP.UseTLS := utUseExplicitTLS;
    oSMTP.AuthType := satNone;
    oSMTP.Connect; // sends EHLO and STARTTLS
    oSMTP.SendCmd('AUTH XOAUTH2 ' + oMail.GetXOAuth2('user@contoso.com'), 235);
    oMessage.From.Address := 'user@contoso.com';
    oMessage.Recipients.EMailAddresses := 'john@example.com';
    oMessage.Subject := 'Hello';
    oMessage.Body.Text := 'Sent with OAuth2.';
    oSMTP.Send(oMessage);
    oSMTP.Disconnect;
  finally
    oMessage.Free;
```

```
    oSMTP.Free;
  end;
end;
```

The strings are built from the current **AccessToken**. When there is no access token, **GetXOAuth2** and **GetOAuthBearer** raise "There is no access token, call Start or Refresh first.".

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: RFC 7628	datatracker.ietf.org/doc/html/rfc7628
Primary standard / spec: XOAUTH2 SASL mechanism	developers.google.com/gmail/imap/xoauth2-protocol
Primary standard / spec: OAuth for IMAP, POP and SMTP	learn.microsoft.com/en-us/exchange/client-developer/legacy-protocols/how-to-authenticate-an-imap-pop-smtp-application-by-using-oauth
Primary standard / spec: RFC 7636	datatracker.ietf.org/doc/html/rfc7636
Primary standard / spec: RFC 8628	datatracker.ietf.org/doc/html/rfc8628
Online help: component page	www.esegece.com/help/sgcWebSockets/ (TsgcMailOAuth2)
Delphi demo project (in the sgcWebSockets package)	`Demos\26.Authentication\06.Mail_OAuth2`
Component page	www.esegece.com/products/websockets/http/mail-oauth2/
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 Mail OAuth2 component shipped with sgcWebSockets Enterprise and the sgcAuth pack. For the full property, method and event reference consult the online help linked above.