

OAuth 2.0 - Mailit 11 General Overview

MailIt supports the highest privacy & security authentication standards. This document explains the general principals of OAuth 2.0 and how to use it with Mailit 11.

Some providers such as Gmail and Outlook, support OAuth 2.0 (OAuth hereinafter), which means that MailIt does not handle your login credentials at all. Instead, OAuth works by approving MailIt as an application, and letting it access your mail, for as long as you would like.

This process involves authorization directly with your provider who already knows your credentials: username and password. Afterwards, MailIt is given a token that you can control and revoke at any given time.

Sometimes, e.g. when your provider has made a change to their security policy, they may ask you to reauthorize your account to refresh your OAuth token.

OAuth is based on an access token concept. When you authenticate yourself (e.g. your Google account) with a third party web application Google authorization server issues an access token to that web application with the approval of the owner. Thus, the web application can use that access token to access your data hosted by the resource server. In the case of Google, your Gmail inbox, contacts, photos etc. are those resources. So any third party application can access those resources, OAuth authorization by Google provided. Hence, OAuth is a simple way to publish and interact with protected resources. It's a save and secure way to grant access to protected resources.

Application Registration

Before using OAuth with your application, you must register your application with the service provider. This is done through a registration form in the “developer” or “API” section of the service’s website. Here, you will provide the following information (and probably details about your application):

- Application Name
- Application Website
- Redirect URI or Callback URL

The redirect URI is where the service will redirect the user after authorizing (or denying) your application, and therefore the part of your application that will handle authorization codes or access tokens.

Client ID and Client Secret

Once your application is registered, the service will issue “client credentials” in the form of a client identifier and a client secret. The Client ID is a publicly exposed string that is used by the service API to identify the registered application, and is also used to build authorization URLs that are presented to users. The Client Secret is used to authenticate the identity of the application to the service API when the application requests to access a user’s account, and must be kept private between the application and the API.

Tokens

When the client application is authorized by the resource owner, the authorization server issues an access token. The client application can use that token to access resource server APIs. For example, a third party application can request an access token from Google server to use Google Contacts API. These tokens are random strings generated by the authorization server.

There are 2 types of tokens:

Access Token: This allows a third-party application to access user data on a resource server. This token is sent by the client as a parameter or as a header in the request to the resource server. It has a limited lifetime, and it must be kept confidential. The authorization server can define the

access token scope. This is a parameter which can be used to limit the access rights for the third party application. The client application should provide the scopes it requires with the request to the authorization server. If the scope is minimum there is a greater chance of being authorized by the user.

Refresh Token: this token is issued with the access token but unlike the latter, it is not sent in each request from the client to the resource server. The client application can simply use a refresh token to renew access token when it expires.

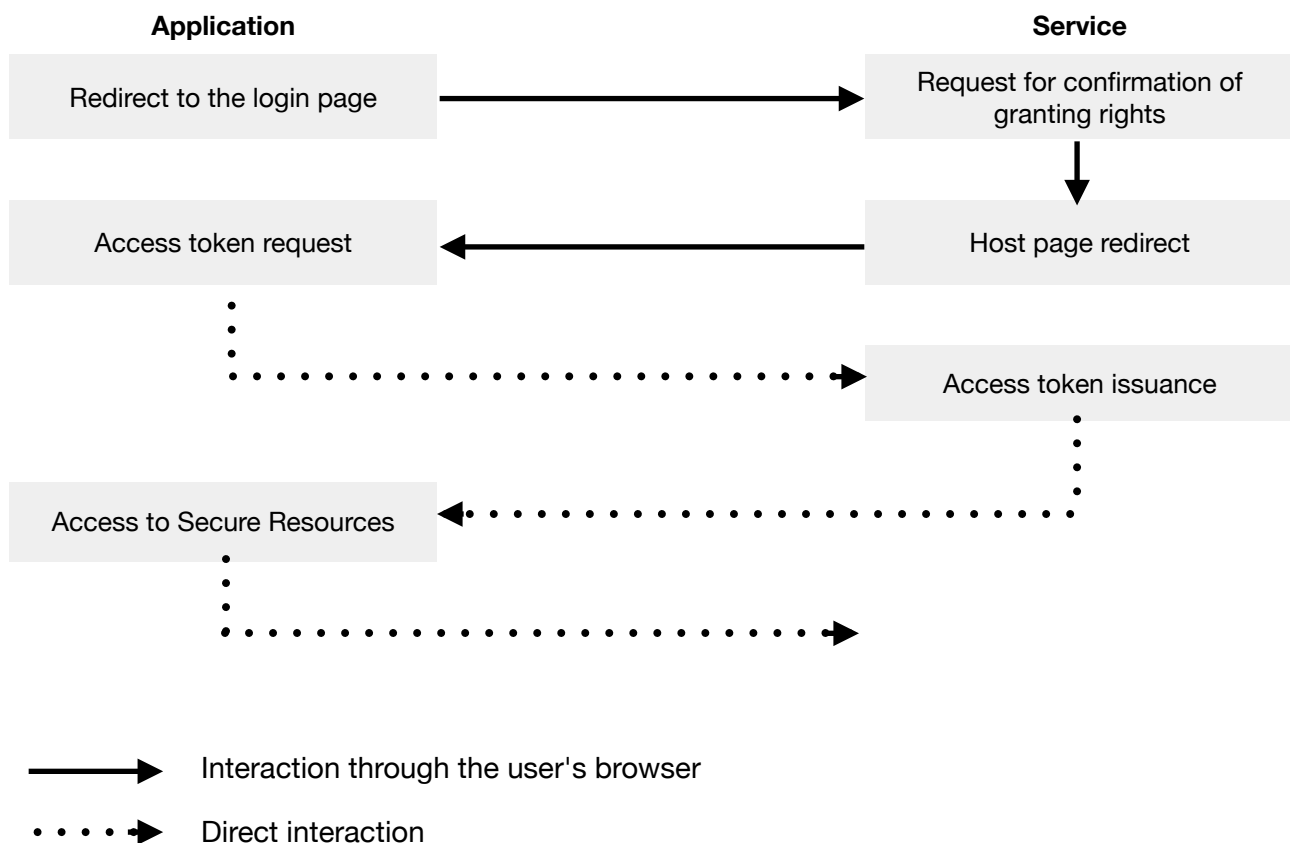
Using a web server in OAuth 2.0 authentication for IMAP / POP / SMTP protocols

OAuth 2.0 uses HTTP and HTTPS for communication between the client and the authorization server because of confidential data for example client credentials, passing between the two applications.

First, the client application requests authorization to access service resources from the user. If the user has not already authenticated with the authorization server, the user will be required to login and consent at the server. After successful authentication, user has to authorize the client at the authorization server. The server responds to the request with an authorization code. The client can then exchange the authorization code with an access token by using client secret. In return, the authorization server will issue an access token and a refresh token in addition to that.

With the acquired access token, the client can send requests to the resource server and access resource server API and user data presenting the access token as the means of authentication.

Authorization for applications



As can be seen from the diagram, in the process of authorization on a remote service using the OAuth mechanism, the service must redirect to our page and also provide additional data such as a temporary key "code" with which we can request access token in the next step.

In order for the service to be able to do the redirection, we need a web server that allows us to accept the redirection and process it. Hence, the Mailit plug-in provides a simple local web server.

Initial setup

First you need to install the plug-in Mailit and additional files for the plug-in, for this, open FileMaker and go to "Preferences" on the "Plug-ins" tab, click the "Reveal Plug-in Folder" button, the "Extensions" directory will open. Copy the Mailit plug-in file and the Mailit Assets folder into the Extensions directory. Restart FileMaker, and again go to "Preferences" on the "Plug-ins" tab, check that the Mailit plug-in is included in the list of plug-ins, and that the plug-in's name is checked.

The next step is to register the application with your service provider.

For the Google Gmail service provider, step-by-step instructions are in the **OAuth 2.0 Google.pdf**. For a Microsoft service provider, a step-by-step guide is located in the **OAuth 2.0 Microsoft.pdf**.

Details: Using a web server in OAuth 2.0 authentication for IMAP / POP / SMTP protocols

The Mailit 11 plugin has a built-in simple static web server. The word "static" means that the server only works with static files, the server does not support working with server languages such as PHP, Ruby, Java, Python, Perl.

To test the examples that are in the Mailit download package, the "Mailit Assets" folder must be copied together with the Mailit plug-in to the FileMaker® Extensions folder. The Mailit plugin automatically looks for the "Mailit Assets" folder within the FileMaker® Extensions folder. If necessary, you can change the name of the "Mailit Assets" folder to any other one and place the directory in any convenient place on your local drive. But when changing the name or location of the folder, you need to pass in the third parameter of the function, "path", the new location and name of the folder in the format of the FileMaker® paths using an absolute path, for example:
filewin:/driveletter:/directoryName/
filemac:/volumeName/directoryName/

When starting the server, the plugin has several reserved addresses:

<http://localhost:1234/> - default server address, you can change it in the settings. HTML, image, text files and directories in the directory "../Mailit Assets/" are available at this address. Only the GET request type is supported.

<http://localhost:1234/oauth/index.html> - used to authenticate with OAuth.

Let us examine the authorization process using the example of "Simple Receiving IMAP.fmp12". For OAuth authorization to work, you need the index.html file which is located in the archive with the plugin along the path "../Mailit Assets/oauth/index.html". The "Mailit Assets" folder is the root folder of the web server.

In the examples, the local web server starts before passing OAuth authorization and stops after passing authorization.

Let's move on to the example.

IMAP Connection Settings

IMAP Server

imap.mycdomain.com

IMAP Server Port

993

SSL/TLS

On

POP Server Timeout

60

IMAP Authentication only

Pref. IMAP Auth.

OAUTH

IMAP User Name

IMAP Password

Leave Messages on Server

On

Delete

Off

After (days)

7

Test Connection

Clear Settings

Last Session Transcript

OAuth 2.0 Client Credentials

OAuth 2.0 Additional settings

Log file

Local WebServer Host

localhost

Local WebServer Port

1234

Start

Stop

OAuth Refresh Token

Authorization URL

Token URL

Scope

Token Expires In

As soon as you open the example, you will be taken to your account layouts. In the "Pref. IMAP Auth. " Select the authentication type "OAUTH", two new tabs "OAuth 2.0 Client Credentials" and "OAuth 2.0 Additional settings" will appear. On the "OAuth 2.0 Additional settings" tab, there are two fields that relate to the local web server, these are the "Local WebServer Host" and "Local WebServer Port" fields. By default, the web server will be available at this address <http://localhost:1234>. You can change the address by specifying other values in the fields. Remember that when creating applications in Google and Microsoft services to pass OAuth authorization, you need to specify the redirect url, it must match the settings specified in the example.

We continue, you need to fill in all the fields for authorization. For more information on how to create an application and get "Client ID" and "Client Secret", see "OAuth 2.0 Google.pdf" and "OAuth 2.0 Microsoft.pdf".

After all the fields are completed, you need to click the "Open OAuth2 WIndow" button, the action will run the "Show OAuth" script in which the local web server will be launched, a new FileMaker window will open with WebView to which the address will be transferred which is based on this calculation:

Expression:

"http://" &
System::Local WebServer Host & ":" & System::Local WebServer Port &

"/oauth/?client_id=" & System::OAuth ID &
"&scope=" & GetAsURLEncoded (System::OAuth Scope) &
"&a_url=" & GetAsURLEncoded (System::OAuth Authorization URL) &
"&time=" & Get (CurrentTimeUTCMilliseconds)

Result:

http://localhost:1234/oauth/?client_id=CLIENT_ID&scope=https:
%2F%2Fmail.google.com&a_url=https:
%2F%2Faccounts.google.com%2Fo%2Foauth2%2Fv2%2Fauth&time=63725674891899

IMAP Connection Settings

IMAP Server

imap.gmail.com

IMAP Server Port

993

SSL/TLS

On

POP Server Timeout

60

IMAP Authentication only

Pref. IMAP Auth.

OAuth

IMAP User Name

dacons.plugin.mailit@gmail.com

IMAP Password

••••••••

Leave Messages on Server

On

Delete

Off

After (days)

7

Test Connection

Clear Settings

Last Session Transcript

OAuth 2.0 Client Credentials

OAuth 2.0 Additional settings

Log file

OAuth Provider

gmail.com

OAuth Client ID

885967412600-5sfosgibb3adg40d7k1jcsvr81lnpi0.apps.googleusercontent.

OAuth Client Secret

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

Use Secret in Request

On

Open OAuth2 Window in FileMaker

Open OAuth2 in External browser

Hint: In order to use OAuth2 authorization you need to create an application.

For the GMail service:
<https://console.developers.google.com/>

For the Outlook or Live service:
<https://portal.azure.com/>

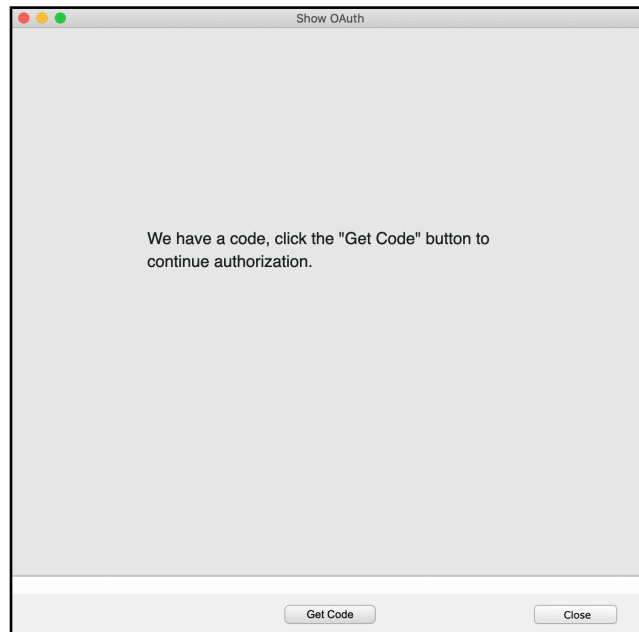
After creating the application, you will be available to "Client ID" and "Client Secret"

We'll parse the address to the "?" Symbol,
http://localhost:1234/oauth/ - the index.html file is located at this address, the file is located in your local file system "../Mailit Assets/oauth/index.html".
Further after the "?" go variables:
client_id - get the client received during the registration process of the application in Google services;
scope - definition of zones on which the client requests access;
a_url - authorization address, made into a variable in order to unify the index.html file and use it for authorization in different services.
time - current time to avoid caching the page in the browser.

©2026 Dacons UG (haftungsbeschränkt)

In the file "index.html" a redirect to the authorization service specified in the variable "a_url" will be generated. After authorization on the specified service, the service will redirect back to the address <http://localhost:1234/oauth/> with additional variables, one of them, the variable "code" with which you can get an "access token".

If everything went well and the service redirected you back with the "code" variable, you will see a message "We have a code, click the "Get Code" button to continue authorization."



When you click the "Get Code" button, the FileMaker script "Get Code" is executed. The script searches for the WebView object on the layout and obtains the current URL. Then, using a simple parser, the value of the variable "code" is contained in the URL. After the value of the variable "code" is successfully received, the "Get Token" script is run by the FileMaker. The script makes a POST request for the "Token URL" specified in the account settings with the transfer variable "code." Upon successful completion of the request, we will receive a response in JSON format with the following data:

- access_token - access key to the service, used as a regular password;
- refresh_token - key for access_token update request;
- expires_in - access_token action time in seconds.

In the examples, in a script called "Connect to the mail server", the automatic update access_token scheme is implemented. Automatic updating will stop working if the OAuth client is denied access to the application settings or the password is changed.