# Authenticating with OAuth v2

⚠ The content of this page applies to all the methods of MailUp REST API except for the Transactional APIs, which do not use OAuth 2.0

## Before you start

MailUp REST API follows OAuth 2.0 specification and authorizes only the applications that specify a pair of valid access keys ("Client ID" and "Client Secret") in the authentication process.

### First, get your API access keys

To get your access keys, you must register your application by following the procedure that is described *here*.

### Basic concepts

Make sure you clearly understand the role of access keys, developer accounts, and MailUp user credentials. These notes should be useful:

- **API access keys** identify a software application (e.g. a plugin for Salesforce that uses MailUp REST API).
  - Access keys are used according to the roles of "client ID" and "client secret" in OAUTH 2.0 specifications.
  - You can use them with any MailUp account, regardless of the account they were created from.
  - By requesting a pair of keys, you register your application. So you have to provide details about it and its author.
- **A developer account** is a free platform that could be requested to MailUp in case your regular account does not let you generate the API access keys.
  - A developer account cannot be converted into a production account and you cannot purchase a subscription plan for it. Use it just for the purposes stated above.
- **User credentials** are the username and the password that identify a user of a specific MailUp account
  - When you start using an application, you are asked to provide the credentials of the MailUp account your application has to be connected to. Only at this moment, you are linking a pair of access keys to a MailUp user
  - Since the allowed relationships between access keys and MailUp users are "many-to-many", you can use the same access keys with several MailUp accounts.

## Let's get started

Once you get the access keys and you have in mind the basic concepts stated above, you are ready to start developing your application. Authentication with OAuth 2.0 is not a walk in the park, the recommended approach is to check out a piece of working code from Samples and Wrappers.

Anyway, if you are not familiar with any of the programming languages of our samples, or if you want to know more, you can have a look at the section below.

---

## Technical reference

### Authorization and authentication

The MailUp REST API uses **OAuth v2** as the authorization and authentication method for validating access to the API resources.

- Your application will be **authorized** to communicate with MailUp using a pair of access keys ("Client ID" and "Client Secret").
- Users of your application will be **authenticated** using their regular MailUp user credentials

The following endpoints have to be used:

- https://services.mailup.com/Authorization/OAuth/LogOn (to start the authorization code grant flow)
- https://services.mailup.com/Authorization/OAuth/Token (to get or refresh an access token)

In case an error occurs during the authorization process, HTTP 400 status code is returned. HTTP 200 or 302 is returned in case of successful authorization. Error descriptions and formats are stated in the OAuth2 protocol documentation, so please refer to it for further details.

Depending on how authorization requests are made, responses can be JSON messages, form-encoded data, or query string parameters. The OAuth v2 protocol better defines the response data format, according to the different flows. Being OAuth v2 a framework, several authorization flows are supported; they are named "Grants" and can be listed as follows:

- Authorization code grant;
- Implicit grant;
- Resource owner password grant;
- Client credentials grant.

MailUp authorization server supports **only** "Authorization code grant flow" and "Resource owner password grant" (aka "Password flow")

## Authorization code grant (recommended)

Authorization code grant (aka "3-legged") is a little bit more complex than the others, but it is robust and safe. In particular, authentication is done on a MailUp page. For your client application, there is no need of storing user credentials, which may become invalid if you change your password. You simply have to call that page passing both the application keys and the URL of your callback page the authentication tokens are returned to. Once you have the tokens you can go on with them. You can use them whenever you need access to an API resource. You just only have to refresh the access token when it expires. There is no need for any further authorization flow unless you have to connect your application to a different MailUp user.

**Get the access token and the refresh token**

```
curl "https://services.mailup.com/Authorization/OAuth/LogOn?
client_id=MYCLIENTID&response_type=code&redirect_uri=MYCALLBACKURL"

Examples:
- MYCALLBACKURL=http://127.0.0.1:8080/rest/index.html
- MYCLIENTID=0a111fe1-aaaa-bbbb-cccc-f33d3d3efcd3


Note: you have to provide a callback page that accepts POST parameters in this format:
        { "access_token":"MYACCESSTOKEN", "expires_in":3600, "refresh_token":"MYREFRESHTOKEN"}
```

## Resource Owner Password Grant

Using the Resource Owner Password Grant (aka "password flow") you can call a resource and immediately get the tokens in the response. There are some warnings you should care about:

- password may change, if you save it on your client your application may stop working when a stored password is no more valid
- if you store the password on your client you are responsible for keeping it safe

**Get the access token and the refresh token**

```
curl -X POST https://services.mailup.com/Authorization/OAuth/Token /
-H "Authorization: Basic BASE64_ENCODED_CLIENTCREDENTIALS" /
-H "Content-Type: application/x-www-form-urlencoded" /
-d "grant_type=password&username=MYUSERNAME&password=MYPASSWORD"


Examples:
- MYUSERNAME=m1234
- MYCLIENTID=0a111fe1-aaaa-bbbb-cccc-f33d3d3efcd3
- MYCLIENTSECRET=f00b000e-aaaa-bbbb-cccc-8f2a92111dde
- BASE64_ENCODED_CLIENTCREDENTIALS=BASE64(MYCLIENTID:MYCLIENTSECRET)

Response:
{
        "access_token":"MYACCESSTOKEN",
        "expires_in":3600,
        "refresh_token":"MYREFRESHTOKEN"
}
```

## Resource access

After passing the authorization process, the client application impersonates a MailUp user. Then, the resources the user is enabled to get or set are accessible by calling related API endpoints. For more detailed information about available resources and related API methods, please refer to the specific documentation of the REST API Resources.

According to the OAuth v2 specification, an access token needs to be provided in the request's HTTP header to access protected information /resources. MailUp API requires using the "Bearer Token". Please check out the example below or refer to the RFC 6750 page for details.

**Resource access by specifying access token**

```
curl -H "Authorization:Bearer MYACCESSTOKEN" /
                -H "Content-Type:application/x-www-form-urlencoded" MYRESOURCEENDPOINT

Examples:
- MYRESOURCEENDPOINT = https://services.mailup.com/API/v1.1/Rest/ConsoleService.svc/Console/List/1/Groups
```

### Request format

These are the available resource endpoints:

- `https://services.mailup.com/API/v1.1/Rest/ConsoleService.svc (email and SMS channels)`
- `https://services.mailup.com/API/v1.1/Rest/MailStatisticsService.svc (email statistics)`
- `https://services.mailup.com/API/v1.1/Rest/PublicService.svc (reserved to applications enabled for account provisioning)`
- `https://services.mailup.com/API/v1.1/Rest/RetailerService.svc (reserved to enabled retailers)`

The structure of the exchanged information is detailed in the **automatic web documentation,** available by appending "/help" to any endpoint's URL address.

Any API resource supports some of the following HTTP verbs:

- **GET**: used to retrieve a resource;
- **POST**: used to create a new resource;
- **PUT**: used to modify an existing resource;
- **DELETE**: used to eliminate an existing resource;

The message request format (XML or JSON) should be declared by specifying the "Content-Type" header field.

### Access restrictions

An application can access an account's resources only if it has a valid pair of API keys and a valid access token is provided.

In addition to these main criteria, the resource access may be subject to one or more of the following constraints (others may be added in the future):

1. *Account or user restrictions*: when accessing from API impersonating a MailUp user, you will find the same restrictions that you have when you log in to the MailUp web application with the same user. E.g. a maximum number of total mailings for trial accounts, possible restrictions on MailUp lists accessible to a user in case of multi-user accounts...
2. *Frequency of calls (Rate Limiting)*: a check is performed on the frequency of the calls by the method, enabling only the calls which fall within a defined range of calls per second (the default value is 5 per second, HTTP 403 is returned when this limit is exceeded)

MailUp reserves the right at any time to limit, suspend or ban any user or client application that may be found abusing the API services.

### Response format

The resource endpoints respond to a request by returning data in either JSON or XML format depending on the value of the "Accept" header parameter specified in the request.

**Returned data format**

```
// JSON format (default)
curl -X GET https://services.mailup.com/API/v1.1/Rest/ConsoleService.svc/Console/Authentication/Info \
  -H 'accept: application/json' -H 'authorization: Bearer MYTOKEN' -H 'cache-control: no-cache' -H
'content-type: application/json'

// XML format
curl -X GET https://services.mailup.com/API/v1.1/Rest/ConsoleService.svc/Console/Authentication/Info \
  -H 'accept: application/xml' -H 'authorization: Bearer MYTOKEN' -H 'cache-control: no-cache' -H
'content-type: application/xml'
```

Please check out the Resources section or refer to the automatic web documentation to learn more about data format.

In case of successful access to the requested resource, an HTTP 200 status code is returned, along with the requested data.
Should an error occur (e.g. caused by malformed request, invalid token, frequency call restrictions, or user access privilege limitations), the following error codes are returned:

- HTTP 400 (Bad request): malformed request or missing parameters. In case of an invalid or malformed authorization token (either refresh token or access token), a proper description message is also returned.
- HTTP 401 (Unauthorized): expired or revoked token.
- HTTP 403 (Forbidden): the application is requesting a resource that is not in the scope of the impersonated user.
- HTTP 429 (Too many requests): it is returned in case of too many requests per second on a single API resource.
- HTTP 500 (Internal error): returned in case of any resource server-internal problem.
- HTTP 503 (Service unavailable): temporary error. The server is currently unable to handle the request due to a temporary overload or scheduled maintenance, which will likely be alleviated after some delay.

OAuth v2 error codes and error descriptions are returned either as a detailed part of the raised fault or as a header parameter as stated in the Bearer token documentation.

## In case of an expired token, you need to refresh it (HTTP 401)

In case of an HTTP 401 error, you need to refresh the authorization token, getting a new one utilizing the refresh token. Please note that also the "refresh token" is renewed with the request and the new value is returned in the response body.

**Refresh the access token**

```
// #1 When resource access fails because token is expired
curl -H "Authorization:Bearer MYACCESSTOKEN" /
                -H "Content-Type:application/x-www-form-urlencoded" MYRESOURCEENDPOINT
Response:{"ErrorCode":"401","ErrorDescription":"Authorization error: Access token is expired","
ErrorName":"Unauthorized","ErrorStack":null}

// #2 First you need to get a new token
curl -X POST -d
"client_id=MYCLIENTID&client_secret=MYCLIENTSECRET&refresh_token=MYREFRESHTOKEN&grant_type=refresh_token"
/
                https://services.mailup.com/Authorization/OAuth/Token
Response:{        "access_token":"MYNEWACCESSTOKEN","expires_in":3600,"refresh_token":"
MYNEWREFRESHTOKEN"}


// #3 Then you can successfully access to the resource by using the new token
curl -H "Authorization:Bearer MYNEWACCESSTOKEN" /
                -H "Content-Type:application/x-www-form-urlencoded" MYRESOURCEENDPOINT
```

## Access errors due to "Contract not signed" (HTTP 403)

Access to MailUp resources may fail if the authenticated user has not signed the "Terms of service" yet. When this happens, you simply have to access to MailUp admin console using the same user and accept terms and conditions that are displayed immediately after user login.

**Refresh the access token**

```
curl -H "Authorization:Bearer MYACCESSTOKEN" /
                -H "Content-Type:application/x-www-form-urlencoded" /
                https://services.mailup.com/API/v1.1/Rest/ConsoleService.svc/Console/List/1/Groups

Response:
{
        "ErrorCode":"403",
        "ErrorDescription":"Authorization error: Contract not signed, please login in console and accept
terms of service.",
        "ErrorName":"Forbidden",
        "ErrorStack":null
}
```

### Access errors due to "Too Many Requests" (HTTP 429)

Access to MailUp resources may fail if the client exceeds the expected rate limit. The returned message shows both the exceeded threshold ("Max") and the number of calls that are received instead ("actual").

**Refresh the access token**

```
curl -H "Authorization:Bearer MYACCESSTOKEN" /
                -H "Content-Type:application/x-www-form-urlencoded" /
                https://services.mailup.com/API/v1.1/Rest/ConsoleService.svc/Console/List/1/Groups

Response:
{
        "ErrorCode":"429",
        "ErrorDescription":"Authorization error (too_many_requests): The call quota is exhausted. Max: 5
calls/second, actual: 9 calls/second, throttling condition expires in: 291 ms, throttling config: 2.",
        "ErrorName":"TooManyRequests",
        "ErrorStack":null
}
```