Genesys Multicloud CX Web-based API Reference

Authentication Overview

5/23/2026

Contents

Genesys Engage APIs rely on the Authentication API for authentication and authorization. If you plan to develop with any of the APIs, you'll need to use the Authentication API as a first step.

The Authentication API enables custom client applications to authenticate users and client applications by generating an authorization token that the client can use to access other APIs.

API Details

Find the API requests, responses, and details here:

- Authentication API

OAuth 2.0

The Authentication API implements authorization flows described in the OAuth 2.0 standard. OAuth is an authorization framework that enables an external application to obtain limited access to an HTTP service, with the consent of the user.

For this purpose, OAuth makes a clear difference between the external application (referred to as the **Client**), the user (referred to as the **Resource Owner**), and the browser (referred to as the **User Agent**).

For more information about the OAuth 2.0 spec, see: RFC 6749 - The OAuth 2.0 Authorization Framework.

Supported OAuth flows

The Authentication API supports the Authorization Code Grant flow defined in OAuth. This is a two-step process:

First, the user authenticates with the Authentication API, using a sign-in form at endpoint `/oauth/authorize`. After a successful authentication, the client application is returned a short-lived authorization code. The authorization code is conveyed to the client application through an HTTP redirect to a specific URI owned by the client application (`redirect_uri`) that needs to be pre-configured (see Requirements for details).

Second, the client application exchanges this code for an access token, by using the `/oauth/token` endpoint, and sending client application credentials. The access token can then be used for making authorized API calls.

The Authorization Code Grant flow is secure and convenient for interactive authentication using web

applications that have a server component. That backend component can store client credentials secretly, and take charge of exchanging the authorization code for an access token.

For more information about the Authorization Code Grant flow, see: RFC 6749 - The OAuth 2.0 Authorization Framework - Authorization Code Grant.

Requirements

You will need the following data for accessing the Authentication API:

- **API Base URL** — Where your client can find the API.
- **API Key** — You must send this as the value for `x-api-key` header of every request.
- **Client credentials** — Credentials for your client application, a **Client ID** and a **Client Password**.
- **User credentials** — Credentials of a user that wants to make use of API functions.

You can request API Key and Client Credentials for your cloud tenant from Genesys Customer Care. From your side, you need to provide the following data:

- **Allowed OAuth Redirect URIs** — During an Authorization Code Grant, the client's browser is redirected to this URI after entering valid credentials. The client application must implement an endpoint at this URI that receives the authorization code and completes the flow. The Redirect URI must adhere to RFC 6749 Section 3.1.2. In particular, it must be an absolute URI that does not include a fragment component.
- **Allowed CORS Origins** (See Cross-Origin Resource Sharing) — If you are calling APIs from a web application, you may need to allow API access to your application, for the browser to permit cross-domain requests.

The Authorization Code Grant flow, step by step

The user navigates to the `/oauth/authorize` endpoint, which displays a sign-in form. It is up to the client to decide how the user navigates to the sign-in page. It can happen by direct navigation, or through an HTTP redirect, or inside an embedded frame or popup window.

The `/oauth/authorize` URL must contain query parameters defined by OAuth. For example:

```
GET /auth/v3/oauth/
authorize?response_type=code&client_id=&state=70db3ab252ead1dd&redirect_uri=https://yourapp.com/
oauthcallback
```

> ### Important
> In this example, the redirect_uri parameter has been provided in clear text for clarity, but it must be URL-encoded.

After a successful sign-in, the Authentication Server conveys the authorization code to your client application. To do this, it redirects the browser to the provided Redirect URI, and adds a query parameter that contains the authorization code. For example, the HTTP redirect response may look like this:

```
HTTP/1.1 302 Found
 Location: https://yourapp.com/oauthcallback&code=ABCXYZ&state=70db3ab252ead1dd
```

After checking the `state` value, your application can exchange the authorization code for an access token, by using the `/oauth/token` endpoint. You must provide client credentials through HTTP basic access authentication. For example:

```
POST /auth/v3/oauth/token
 Authorization: Basic
 Content-Type: application/x-www-form-urlencoded

 grant_type=authorization_code&code=ABCXYZ
 &redirect_uri=https://yourapp.com/oauthcallback
```

Again, the `redirect_uri` must be URL-encoded.

Finally, the Authentication API returns an access token to your client application, inside a JSON response. For example:

```
HTTP/1.1 200 OK
 Content-Type': application/json;charset=utf-8
 Cache-Control': no-store
 Pragma: no-cache

 {'access_token': '1234abcd-5569-4fa9-bfad-12345678abcd',
  'expires_in': 43199,
  'refresh_token': '456789ef-3dce-4c82-bc1f-987654324e87',
  'scope': '*',
  'token_type': 'bearer'}
```

How to use the access token

Once your client application has the access token, it can access the APIs by including the token inside the `Authorization` header of HTTP requests to API endpoints.

```
Authorization: Bearer 1234abcd-5569-4fa9-bfad-12345678abcd
```

An access token is valid for a certain time. Once the access token expires, if your application obtained a refresh token, your application may use it to obtain a new access token, as described in OAuth 2.0 - Refreshing an Access Token.

> ### Important
> An access token is given a certain lifetime when generated, which may be affected by security settings or the user's actions. This means a client application must not rely on the expires parameter, and must provide for dynamically reacquiring an access token.

If an application token is expired or invalid, the API returns the HTTP status code 401. This lets the application know it must get a new access token.

Troubleshooting authentication issues

Review the error messages below to see solutions for common troubleshooting scenarios.

```
HTTP/1.1 403 Forbidden
 {'message': 'Forbidden'}
```

If you get this response from any API endpoint, you are probably using an invalid API Key in your `x-api-key` request header.

```
HTTP/1.1 401 Unauthorized
 {'error': 'Unauthorized', 'error_description': 'Bad credentials'}
```

If you get this response from the `/oauth/token` endpoint, you are probably using invalid client credentials. Check you Client ID and Client Password.

```
HTTP/1.1 400 Bad Request
 {'error': 'invalid_grant', 'error_description': 'Bad credentials'}
```

If you get this response from the `/oauth/token` endpoint, you are probably using invalid user credentials. Check you user name and password.

Client libraries

Genesys also offers client libraries for the Authentication API in both Node.js and Java. These libraries may help you start up quicker, as they take care of a lot of the supporting code needed to make the proper HTTP requests and handle HTTP responses.

What About Basic Authentication?

HTTP Basic access authentication has been intentionally disabled. While basic authentication is the simplest way to authenticate a user, it does not provide the security benefits of OAuth. There are a few key security concerns with basic auth:

- A hash of the user's username and password are sent with every request; this increases the exposure of sensitive data.
- The basic auth hash value never expires, so anyone intercepting that value will gain access to the account until the user's password is changed.
- When using basic authentication, the user must provide their account credentials directly to the application, which means that something other than the user themselves and the Authentication API have direct access to their account credentials.