Authorization code workflow for Stock

Table of Contents

Overview **1**
Basics of OAuth authentication **2**
Getting started **3**
Setting up the environment **3**
Platform-specific instructions for creating an HTTPS server 4
Adobe I/O application integration **4**
Auth code client-server process **5**
Signing in **5**
Renewing your login 8
Signing out **9**
Caveats—please read! **10**

This document describes the steps to authenticate users for Adobe Stock using the *authorization code* OAuth flow. This guide focuses only on the authentication portion rather than usage of the Stock API, which is covered in other guides. From that sense, this guide can also be applied to authentication for any application that consumes a Creative Cloud service.

Important See the caveats and warnings listed at the end of the document before integrating this solution.

Overview

As background, the goal of this workflow is to enable third-party applications which use the Adobe Stock API to allow their end-users to sign in using Adobe's IMS (Identity Management Service), which can either authenticate users directly, or redirect them to their corporate SSO provider for authentication, and through this login, allow them to use the Stock entitlements they have access to.

The Adobe Stock API supports different models of the OAuth 2.0[1] authentication scheme, which allows a third-party application limited access to a protected HTTP service. This document describes the

[1] https://tools.ietf.org/html/draft-ietf-oauth-v2-31

authorization code model,[1] which is more secure than the Creative SDK *implicit grant* model described in a separate document, because the access token is not shared with the front-end JavaScript, which could potentially be viewed by others.

When deciding whether to use the Creative SDK (CSDK) implicit grant method or the authorization code ("auth code") method described in this document, there are a few factors to consider:

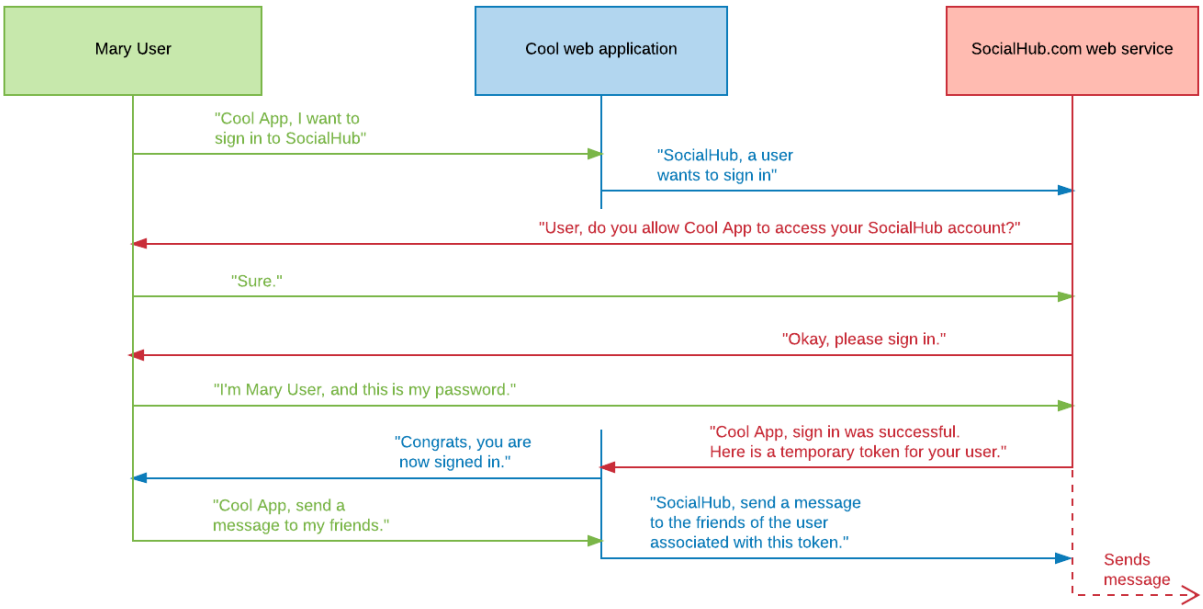
1. **Ease of implementation.** If this is your main concern, choose the CSDK. The CSDK has libraries that automate the process, such that your app simply needs to call login and logout method from your front-end application, and the CSDK will handle all parts of the login. By comparison, the auth code workflow requires you to build custom code to call the IMS endpoints directly and handle their responses.
2. **Security.** The auth code method is inherently more secure, because protected calls to IMS are made "behind the scenes," on your backend. Once the user is signed in, there is no need to expose the access token to the front-end code, which makes it less susceptible to attacks.
3. **Application architecture constraints.** If the app only resides in the browser and does not have access to a server, CSDK is the only practical method. On the other hand, if the app cannot handle client-side redirects, then the auth code method may be more suitable. In either case, the user must be able to sign into Adobe via its website, similar to the sign-in method used by Google and Facebook.

Basics of OAuth authentication

OAuth allows end-users to sign in directly to the service they want to use from within the convenience of an intermediate application. The app acts as "middle-man" between the user and the web service, but importantly does not handle user names and passwords. Instead, the web service redirects the user to the website for login, and then gives back a token to the application allowing it to speak on behalf of the user—assuming the user gives the application permission.

A very basic representation of how this works is below. For specific details on how Adobe implements OAuth in its workflows, see the documentation on Adobe I/O.[2]

[1] https://tools.ietf.org/html/draft-ietf-oauth-v2-31#section-1.3.1
[2] https://www.adobe.io/authentication/auth-methods.html#!adobeio/adobeio-documentation/master/auth/OAuth2.0Endpoints/web-oauth2.0-guide.md



The basic flow is illustrated above, and is similar to how a third-party application would enable its end-users to sign in directly to Adobe to use their Stock entitlements. From this flow, it is assumed the application is able to:

1. Interact with the web service's identity provider.
    - This is handled by direct server-to-server calls with Adobe IMS.
2. Provide a secure redirect that will be the location that the web service sends the user after authentication is successful. The OAuth specification requires that the flow begins and ends within the app hosted on your server, whether the redirect is an HTML page or a server-side script.
    - In the proposed workflow below, the redirect will be a server-side script.
3. Receive a temporary access token from the web service after the user has signed in, and send that token back to the web service with every request from the user.
    - Once the user is signed in, IMS will provide this token. The application will need to send the token in the header of each HTTP request.

Getting started

Setting up the environment

To test and integrate with the auth code method, you must create a secure (HTTPS) server. This is required by the auth code method, which will redirect traffic from Adobe's sign in page back to your server, but only if your page is hosted in a secure location. Also, if you prefer not to use front-end Ajax to communicate with the Adobe Stock API, you will need server support to handle these requests.

For basic testing, a simple option is to use Node.js or Python; otherwise, Apache provides a robust set of tools for web hosting. In each case, you will need to generate or purchase a public key certificate to complete the setup. For more information on working with certificates, see the earlier link to Adobe I/O.

Platform-specific instructions for creating an HTTPS server

1. Node.js:[1] https://www.whatsthatlambda.com/nodejs/creating-an-https-server-with-nodejs-and-express
2. Python:[2] https://anvileight.uk/blog/2016/03/20/simple-http-server-with-python/
3. Apache:[3] https://www.digicert.com/ssl-certificate-installation-apache.htm

Adobe I/O application integration

To use the Adobe Stock API or auth code method, you will need to create an application key on the Adobe I/O Console. Adobe I/O will whitelist this key and permit your application access to the APIs. It also issues and validates OAuth claims.

1. Access the Adobe I/O Console here: https://console.adobe.io.
    - If you do not already have an Adobe ID, you will need to create one (for free).
2. Click the **New Integration** button.
3. Select the following items, clicking **Continue** each time:
    - **Access an API** > **Creative Cloud / Creative SDK**[4] > **New integration**
4. This opens a screen where you will enter your integration details.
    - **Name**: Your application's name. This will not be sent in your API requests; however, a good practice might be to give it the same web-friendly name you will be using later when sending the required X-Product header (see *Application flow details*, below).
    - **Description**: E.g., "Integration of Stock API with MyWebsite.com."
    - **Platform**: Choose Web. This document assumes you are authoring a web/browser-based application as opposed to a native iOS or Android application.
    - **Default redirect URI**: As mentioned earlier, this is the URL of the page or script (usually at the root of your web app) which Adobe will access during the authentication process. It must be hosted on a secure (HTTPS) server, even if it is only a localhost instance

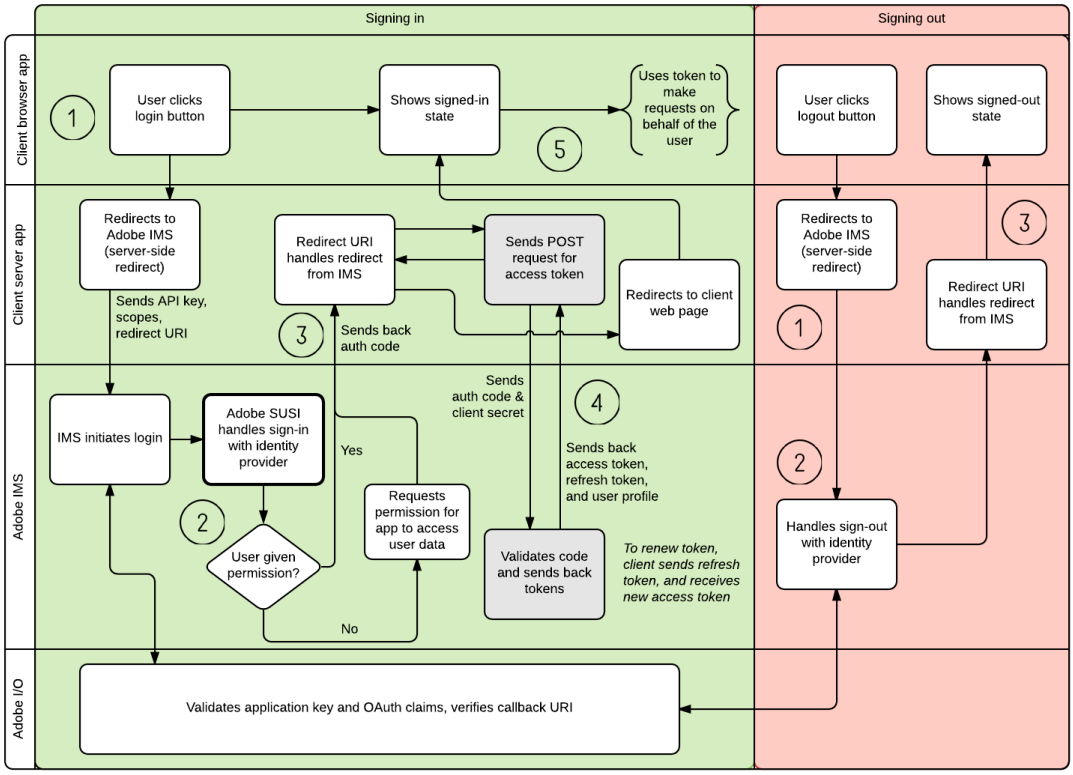
[1] https://nodejs.org/en/
[2] https://www.python.org/
[3] https://www.apache.org/
[4] The CSDK provides the correct scopes required by the auth code workflow. *See notes at end of paper.*

- If you do not have this address yet, you can use any URL address (e.g., `https://mysite.com/redirect.html`.) You will need to change this later for your application to work, however.

- **Redirect URI pattern**: This is a URI path (or comma-separated list of paths) to which Adobe will attempt to redirect when the login flow is complete. It must be within your application domain, and is typically the root. You must escape periods (.) with `\\`.
  - Ex: `https://mysite\\.com/`

1. Once saved, the I/O Console will generate several pieces of information you will need later.
   - Copy everything in the **Client Credentials** section (including the Client Secret, which you must safeguard like your private key), and store in a secure location.

Auth code client-server process



Before you start creating your application, it is important to understand how your client-side app interacts with your server-side app, and how it in turn communicates with Adobe IMS and Adobe I/O.

Signing in

1. The process begins when the user clicks the login button in the client browser app, which calls an endpoint on the client server app, which redirects to the IMS authorization endpoint. This notifies Adobe IMS to start the sign-in process. It's recommended that your app proxies communication with IMS, so that your front end does not expose any secure information.

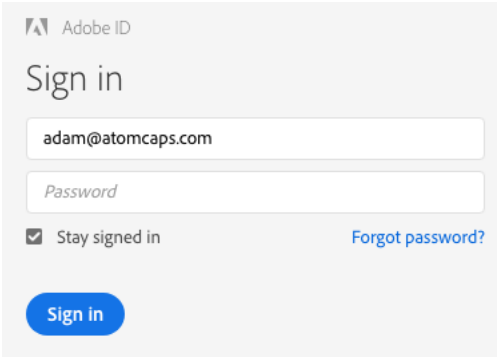
- IMS URL
  - `https://ims-na1.adobelogin.com/ims/authorize`
- Parameters
  - `client_id`: API key obtained from Adobe I/O
  - `scope`: **`openid,creative_sdk`**
  - `redirect_uri`: Path needs to match the redirect in the Adobe I/O integration
  - `response_type`: **`code`**

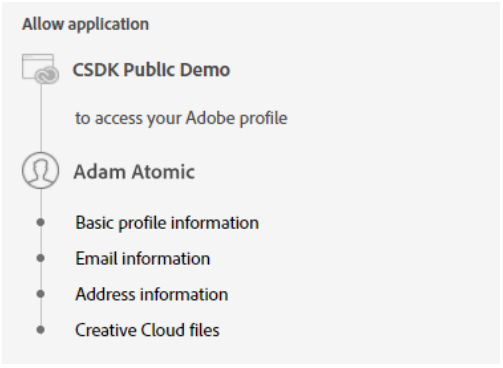
Server script redirects to IMS authorization endpoint

```
GET /auth/signin HTTP/1.1
Host: localhost:8443

HTTP/1.1 302 Found
Location: https://ims-na1.adobelogin.com/ims/authorize
  ?client_id=3a67c...
  &redirect_uri=https://localhost:8443/auth/token
  &scope=openid,creative_sdk
  &response_type=code
```

2. Adobe IMS will redirect to the familiar Adobe sign-in page, called "SUSI" ("Sign Up/Sign In.")
   - Depending on the user's email address, authentication will be handled either by Adobe's identity provider, or the Enterprise identity provider of the user's parent organization.
   - If the user has not granted permission, IMS will first ask permission from the user to allow the application to access the user's information. The name of the app making the request will be the one set in the Adobe I/O Console, earlier.





3. If the user has signed-in successfully, Adobe IMS will redirect the browser back to the client redirect URI, with an authorization code in the query string.
   - It is recommended that the redirect location be a server script and not a web page, since this code is part of the authentication sequence and should be kept secure.

IMS redirects back to the application

```
GET /auth/token?code=eyJ4NXU...vkCnh9Q
HTTP/1.1
Host: localhost:8443
```

4. Once the auth code is received, the server app will send a separate POST request to IMS, providing the API key, auth code and client secret (obtained earlier from Adobe I/O). The response will be both an access token and a refresh token. In addition, the response will include the user profile, which is a convenience method since a common workflow is to immediately get the user profile once the user is signed in.
    - IMS URL
        - `https://ims-na1.adobelogin.com/ims/token`
    - Parameters
        - `grant_type`: **`authorization_code`**
        - `client_id`: API key obtained from Adobe I/O
        - `client_secret`: Obtained from Adobe I/O
        - `code`: Code sent by IMS in redirect in step #3.

Send IMS POST access token request

```
POST /ims/token HTTP/1.1
Host: ims-na1.adobelogin.com
Content-Type: application/x-www-form-urlencoded
  grant_type=authorization_code
  &client_id=3a67c...
  &client_secret=12e7...
  &code=eyJ4NXU...vkCnh9Q
```

IMS responds with access and refresh tokens, and user profile

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
{
    "access_token": "eyJ4NXU...qOk8-DA",
    "refresh_token": "eyJ4NXU...ZoQP_5A",
    "sub": "5BEB2BBC46CDB90599201549@AdobeID",
    "address": {
        "country": "US"
    },
    "email_verified": "true",
    "name": "Adam Atomic",
    "token_type": "bearer",
    "given_name": "Adam",
```

```
    "expires_in": 86399985,
    "family_name": "Atomic",
    "email": "adam@atomcaps.com"
}
```

5. Now the user is signed in, and the server app can notify the front-end to show the signed-in state. From here, the app will use the access token with every API request made by the user (the token is optional for Stock search requests, but required for licensing requests).
    - For example, the Member/Profile Adobe Stock request requires an access token (passed in the Authorization "Bearer" header). Refer to the Stock Licensing API reference for details.[1]

Send authenticated Member/Profile request to Stock API

```
GET /Rest/Libraries/1/Member/Profile?content_id=117487990&
locale=en_US HTTP/1.1
Host: stock-stage.adobe.io
X-Product: IMSDemo
x-api-key: 3a67c...
Authorization: Bearer eyJ4NXU...qOk8-DA
```

Member/Profile sample response

```
{
    "available_entitlement": {
        "quota": 85,
        "license_type_id": 15,
        "has_credit_model": true,
        "has_agency_model": false,
        "is_cce": true,
        "full_entitlement_quota": {
            "credits_quota": 75,
            "image_quota": 85
        }
    }, ...
}
```

Renewing your login

One of the major features of the auth code workflow is the ability to renew or "refresh" your access periodically without needing to sign-in again. This is done using the refresh token. While the access token is designed to expire over a short amount of time (default is 24 hours), the refresh token lasts for up to

[1] https://www.adobe.io/apis/creativecloud/stock/docs.html#!adobe/stock-api-docs/master/docs/getting-started/apps/06-licensing-assets.md

two weeks, by default. Therefore, within that time, you can ask Adobe IMS to issue a new access token, and all that is required is a single API call that can occur behind the scenes.

6. To request a refresh token, the process is almost identical to requesting the access token (step #4, above), except changing the value of `grant_type` and replacing `code` with a `refresh_token` parameter:
    - IMS URL
        - `https://ims-na1.adobelogin.com/ims/token`
    - Parameters
        - `grant_type`: **`refresh_token`**
        - `client_id`: API key obtained from Adobe I/O
        - `client_secret`: Obtained from Adobe I/O
        - `refresh_token`: *Original refresh token* sent by IMS in step #4.

Send IMS POST refresh token request

```
POST /ims/token HTTP/1.1
Host: ims-na1.adobelogin.com
Content-Type: application/x-www-form-urlencoded
  grant_type=refresh_token
  &client_id=3a67c...
  &client_secret=12e7...
  &refresh_token=eyJ4NXU... Ps1hKQug
```

Signing out

1. Your app provides a logout button. When the user clicks it, the front-end redirects to a logout endpoint on your server app, which calls the logout endpoint on IMS. Like the sign-in process, best practice is to let the server app call IMS, since the access token must be passed as part of the request.
    - IMS URL
        - `https://ims-na1.adobelogin.com/ims/logout`
    - Parameters
        - `access_token`: Last access token obtained from login
        - `redirect_uri`: Path needs to match the redirect in the Adobe I/O integration

Redirect to IMS logout endpoint

```
GET /auth/signout HTTP/1.1
Host: localhost:8443

HTTP/1.1 302 Found
```

```
Location: https://ims-na1.adobelogin.com/ims/logout
  ?access_token=eyJ4NXU...qOk8-DA
  &redirect_uri=https://localhost:8443/auth/token
```

2. When the process is finished on the Adobe IMS side, IMS redirects the browser back to the redirect URI, and your app can notify the front end so the UI can be updated to show the signed-out state.

Logout response from IMS

```
GET /ims/logout_response
  ?redirect_uri=https://localhost:8443/auth/token
  &client_id=3a67c... HTTP/1.1
Host: ims-na1.adobelogin.com

HTTP/1.1 302 Found
Content-Type: text/html;charset=UTF-8
Location: https://localhost:8443/auth/token
```

Caveats—please read!

While this workflow will accomplish your goals, be aware of the following important notes:

- **This is an interim solution.**

Adobe will be releasing an SDK in 2019 which will automate this workflow using multiple programming languages like NodeJS, PHP and Java. At that time, it will no longer be necessary (or desirable) to access the IMS endpoints directly. IMS endpoints and the API syntax described in this paper may change in the future, so once the new SDK is available, all customers and partners should switch to it immediately. In the future, this workflow will only be supported by Adobe if using the SDK method.

- **Overage licensing must be blocked.**

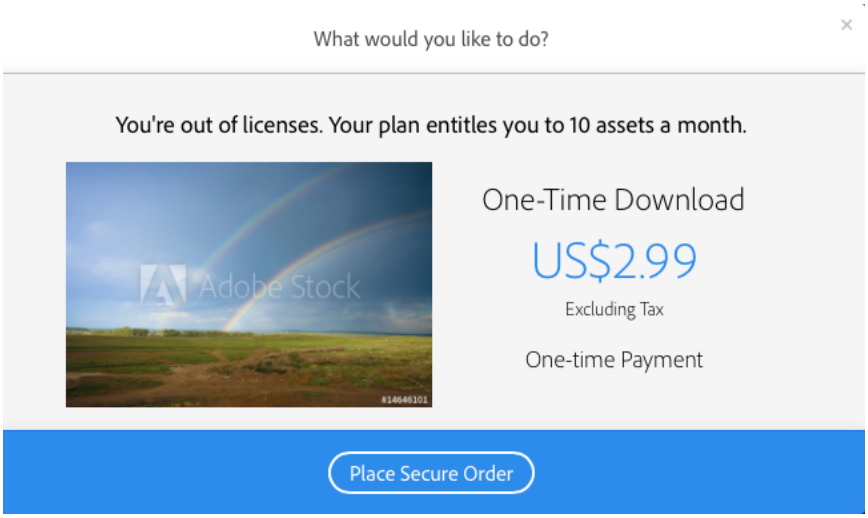
As described in the Stock Licensing API documentation,[1] it is possible for customers to enter a state called "Overage." This happens when a customer has used all their licenses or credits, but has a purchase instrument on file (such as a credit card or PayPal account.) The Member/Profile response will indicate that licensing is possible because of the overage state.

Member/Profile response for Overage state

```
  "purchase_options": {
    "state": "overage",
    "requires_checkout": false,
    "message": "Would you like to license the image for $2.99?"
  },
```

[1] https://www.adobe.io/apis/creativecloud/stock/docs.html#!adobe/stock-api-docs/master/docs/api/12-licensing-reference.md

When this situation occurs on the Adobe Stock website, users are presented with a dialog and taken through a checkout workflow (see below). But in the API, there is no mechanism to enforce this.



As a result, it is possible in the API for an end user to be charged extra without their approval. They will receive an email after the fact, but would not be alerted before the charge happens.

Net, when using the API, it is necessary for applications to **block licensing** when a customer is in the Overage state. In this case, the customer should either be directed to go to the Adobe Stock website to complete the transaction, or simply told they do not have credits or licenses available.

Otherwise, your application can cause legal issues with your customers for charging them without their knowledge.

- **Use of Creative SDK instead of Adobe Stock application integration.**

As mentioned in the documentation, it is necessary to create a new Creative SDK application integration instead of an Adobe Stock integration. The main reason is that only the Creative SDK integration has the scopes required for a successful sign in.

But an equally important reason is that if you use only the Adobe Stock integration method, your users will not be asked to give consent (see the Signing -in workflow, above.) And furthermore, they would not be able to revoke consent from Adobe.com. Again, this can cause legal/privacy issues if your users cannot block your app from accessing their data, so only the Creative SDK scope ensures this permission is given.

- **(Currently) no support for Social sign-in.**

There is currently (as of April 2018) a bug in the sign in process that shows social sign in options from Facebook and Google, but these buttons do not work. The buttons work on first-party integrations created by Adobe, but do not work in third-party integrations.

Consequently, it's recommended that you open a Zendesk ticket with Adobe I/O to disable social sign in, to prevent confusion by your users.

- **Enterprise login disabled by default.**

Use of the Creative SDK was designed for use by Creative Cloud Individual and Team users, not by enterprise accounts. Subsequently, if a user signs in to the app and tries to access their Adobe Stock Enterprise account, they will be blocked.

If your application needs to support Adobe Enterprise account users, please contact Grp-AdobeStockPartnerships@adobe.com to discuss your use case.

Adobe Systems Incorporated
345 Park Avenue
San Jose, CA 95110-2704
USA
www.adobe.com

5/15