Authentication

The Instagram API uses the OAuth 2.0 protocol for simple, but effective authentication and authorization. OAuth 2.0 is much easier to use than previous schemes and developers can start using the Instagram API almost immediately. The one thing to keep in mind is that all requests to the API must be made over SSL (http**s**:// not http://).

Do you need to authenticate?

The Instagram API requires authentication - specifically requests made on behalf of a user. Authenticated requests require an **access_token**. These tokens are unique to a user and should be stored securely. Access tokens may expire at any time in the future.

Receiving an access_token

In order to receive an **access_token**, you must do the following:

1. Direct the user to our authorization url.
   - If the user is not logged in, they will be asked to log in.
   - The user will be asked if they would like to grant your application access to her Instagram data.
2. The server will redirect the user in one of two ways that you choose:
   - **Server-side flow (recommended):** Redirect the user to a URI of your choice. Take the provided **code** parameter and exchange it for an access_token by POSTing the code to our access_token url.
   - **Implicit flow:** Instead of handling a code, we include the access_token as a fragment (#) in the URL. This method is less secure, but allows applications without any server component to receive an access_token.

Important

Even though our access tokens do not specify an expiration time, your app should handle the case that either the user revokes access, or Instagram expires the token after some period of time. If the token is no longer valid, API responses will contain an "error_type=OAuthAccessTokenException". In this case you will need to re-authenticate the user to obtain a new valid token.
In other words: **do not** assume your access_token is valid forever.

Server-side (Explicit) Flow

Using the server-side flow is quite easy. Simply follow these steps:

Step One: Direct your user to our authorization URL

```
https://api.instagram.com/oauth/authorize/?client_id=CLIENT-ID&redirect_uri=REDIRECT-URI&response_type=code
```

Note: You may provide an optional **scope** parameter to request additional permissions outside of the "basic" permissions scope. Learn more about scope.

Note: You may provide an optional **state** parameter to carry through a server-specific state. For example, you can use this to protect against CSRF issues.

At this point, we present the user with a login screen and then a confirmation screen where to grant your app access to her Instagram data.

Step Two: Receive the redirect from Instagram

Once a user authorizes your application, we issue a redirect to your redirect_uri with a **code** parameter to use in step three.

```
http://your-redirect-uri?code=CODE
```

Note that the host and path components of your redirect URI must match exactly (including trailing slashes) your registered redirect_uri. You may also include additional query parameters in the supplied redirect_uri, if you need to vary your behavior dynamically. Examples:

REGISTERED REDIRECT URI	REDIRECT_URI PARAMETER PASSED TO /AUTHORIZE	VALID?
http://yourcallback.com/	http://yourcallback.com/	yes

http://yourcallback.com/	http://yourcallback.com/?this=that	yes
http://yourcallback.com/?this=that	http://yourcallback.com/	no
http://yourcallback.com/?this=that	http://yourcallback.com/?this=that&another=true	yes
http://yourcallback.com/?this=that	http://yourcallback.com/?another=true&this=that	no
http://yourcallback.com/callback	http://yourcallback.com/	no
http://yourcallback.com/callback	http://yourcallback.com/callback?type=mobile	yes

If your request for approval is denied by the user, then we will redirect the user to your **redirect_uri** with the following parameters:

- **error**: access_denied
- **error_reason**: user_denied
- **error_description**: The user denied your request

```
http://your-redirect-uri?
error=access_denied&error_reason=user_denied&error_description=The+user+denied+your+request
```

It is your responsibility to fail gracefully in this situation and display a corresponding error message to your user.

Step Three: Request the access_token

Now you need to exchange the **code** you have received in the previous step for an access token. In order to make this exchange, you simply have to POST this code, along with some app identification parameters, to our access_token endpoint. These are the required parameters:

- **client_id**: your client id
- **client_secret**: your client secret
- **grant_type**: **authorization_code** is currently the only supported value
- **redirect_uri**: the redirect_uri you used in the authorization request. Note: this has to be the same value as in the authorization request.
- **code**: the exact code you received during the authorization step.

This is a sample request:

```
curl -F 'client_id=CLIENT_ID' \
-F 'client_secret=CLIENT_SECRET' \
-F 'grant_type=authorization_code' \
-F 'redirect_uri=AUTHORIZATION_REDIRECT_URI' \
-F 'code=CODE' \
https://api.instagram.com/oauth/access_token
```

If successful, this call will return a neatly packaged OAuth Token that you can use to make authenticated calls to the API. We also include the user who just authenticated for your convenience:

```
{
    "access_token": "fb2e77d.47a0479900504cb3ab4a1f626d174d2d",
    "user": {
        "id": "1574083",
        "username": "snoopdogg",
        "full_name": "Snoop Dogg",
        "profile_picture": "..."
    }
}
```

Client-Side (Implicit) Authentication

If you are building an app that does not have a server component (a purely javascript app, for instance), you will notice that it is impossible to complete step three above to receive your access_token without also having to store the secret on the client. You should **never** pass or store your client_id secret onto a client. For these situations there is the Implicit Authentication Flow.

Step One: Direct your user to our authorization URL

```
https://api.instagram.com/oauth/authorize/?client_id=CLIENT-ID&redirect_uri=REDIRECT-
URI&response_type=token
```

At this point, we present the user with a login screen and then a confirmation screen where they grant your app's access to their Instagram data. Note that unlike the explicit flow the response type here is "token".

Step Two: Receive the access_token via the URL fragment

Once the user has authenticated and then authorized your application, Instagram redirects them to your redirect_uri with the access_token in the url fragment. It will look like this:

```
http://your-redirect-uri#access_token=ACCESS-TOKEN
```

Simply grab the access_token off the URL fragment and you're good to go. If the user chooses not to authorize your application, you'll receive the same error response as in the explicit flow