Using OAuth 2.0 with Kiind or Giftbit

Document version 1.1.1, October 2014.

Changelog

v1.0 - August 2014 - initial release
v1.1 - October 2014 - added PAYMENT scope and support for Giftbit
v1.1.1 - October 2014 - clarified expiry lengths of API and Refresh tokens

Overview and Abstract

The Kiind OAuth 2.0 endpoints allow your users to connect their Kiind gifting or Giftbit merchant accounts to your application in a secure manner, after which your application can take actions on their behalf (such as sending gifts or accepting gifts as payment) via our REST APIs.   This is conceptually the same as a user connecting their FaceBook or Twitter account to your application (after which your application could post/tweet on their behalf). This document is relevant for two types of integrations:

1) Allowing your users to send Kiind gifts seamlessly from within your application.
2) Accepting Giftbit gift cards as a payment type on behalf of your merchants in your checkout or point of sale software

This document provides the technical information necessary to implement OAuth 2.0 authentication with Kiind within your application.

Note: If you are connecting directly to our API to send or accept your own gifts, you will be issued an API Token and do not need to use the OAuth endpoints; this document is not relevant to your usecase.

Getting API Access

If you have not already done so, you will need to submit some initial details about your use case, after which the Kiind team will supply you with your sandbox credentials and everything you need to know to get going in our test environment, which is where you will build out and test your integration.

Sample Application and Code

We have a simple Grails application that you can download the source code and run with your Kiind testbed credentials that gives a working example of the authentication flows and API calls. This sample application contains all the pieces and API calls to Kiind you will need to build put in your own application and we highly recommend using it as a reference for building and debugging your own work. Download it here

Types of OAuth flows supported

We support three types of authentication flows defined in the OAuth 2.0 specification:

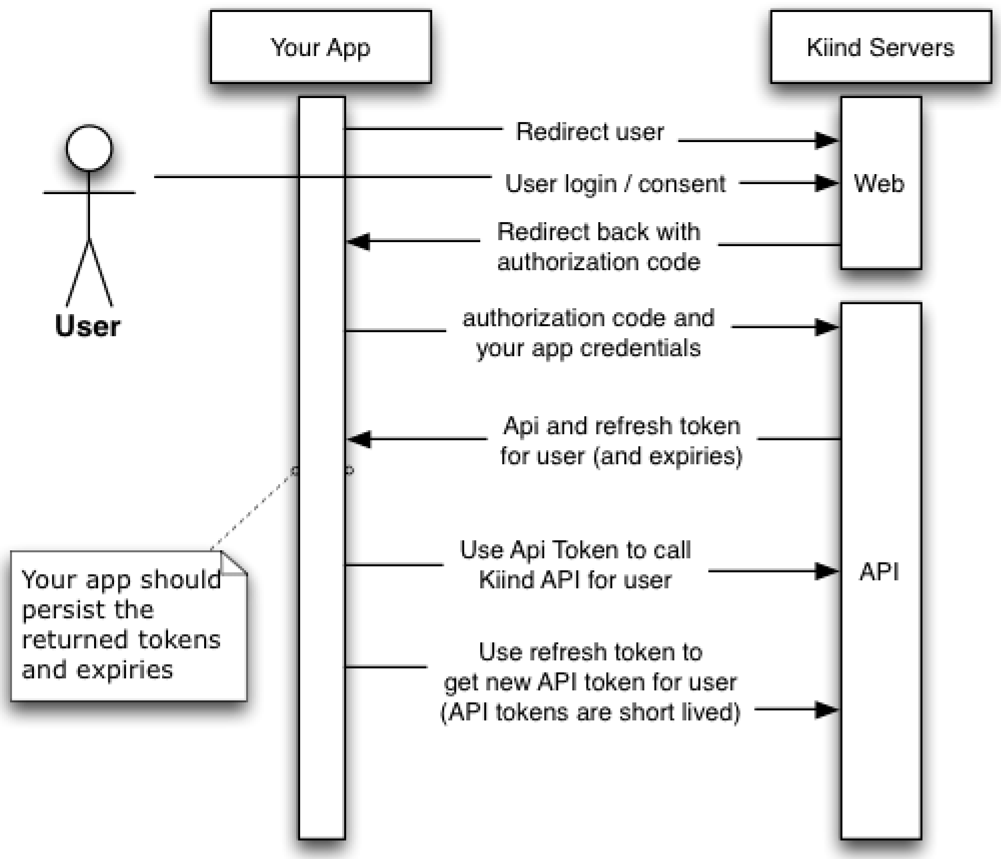
1. Authorization Code Grant Flow - this flow *must* be used unless there is a technical reason why it cannot be used. It is applicable for the majority of web based applications and the flow described in this document.
2. Implicit Grant - this flow is applicable in situations where it is not possible to securely authenticate your client application, such as public/JavaScript clients.
3. Resource Owner Password Credentials Grant - this flow is applicable if you don't have the ability to redirect the user for authentication, such as native mobile/tablet applications.

The API team is happy to discuss your implementation with you to ensure you implement the correct flow. If it is determined you need to use a flow other than Authorization Code Grant, we will supply supplementary documentation.

Authorization Code Grant Flow

This is the most secure way for your users to allow your application to access their Kiind accounts, as Kiind handles all user authentication and the user never needs to provide their Kiind credentials to your application. It is this flow that Kiind supports and is described in this document, and it directly implements the 'Authorization Code Grant' flow from the OAuth 2.0 specification.

The below diagram shows the request and response flow between your application and the Kiind web and API servers. Each step is explained in detail below with call examples


Your App
Kiind Servers
User
Redirect user
User login / consent
Web
Redirect back with authorization code
authorization code and your app credentials
Api and refresh token for user (and expiries)
Your app should persist the returned tokens and expiries
Use Api Token to call Kiind API for user
API
Use refresh token to get new API token for user (API tokens are short lived)

Step 1: Redirecting the user and getting the authorization code

The authorization sequence begins when your application redirects the user's browser to Kiind's user authentication endpoint with appropriate parameters that identify your application and your redirect URL. Upon successful user authentication, Kiind will redirect the user back to your supplied redirect URL with an authorization code in the query string. It is this authorization code that you will then trade for an actual access and refresh token (explained in later section).

Note that the redirect_url must *exactly* match one that you have registered for your application in your Kiind developer portal (see below).

Kiind developer portal (you must have an API enabled account in the respective environmen visit http://info.kiind.me/api to get started if you do not)
PRODUCTION
https://www.kiind.me/devportal
SANDBOX
https://testbed.kiind.me/devportal

The rest of this section shows how to build up the user authentication URL, and what to expect in the query string when Kiind redirects the user back to your redirect URL.

User authentication web endpoint
PRODUCTION
https://www.kiind.me/oauth/userlogin
SANDBOX
https://testbed.kiind.me/oauth/userlogin

Building the URL

The following table shows the expected query string parameters. All parameters are required unless marked as optional. Note that all values must be URL Encoded.

Parameter	Values	Description
response_type	code	Tells Kiind to return an authorization code when redirecting the user back to your application after successful authentication
client_id	The application ID that you have registered with Kiind in your development console	Identifies you as the application making the request. The value passed in this parameter must exactly match the value shown in the developer console and is case sensitive.
redirect_uri	One of the URLs that you have registered with Kiind in your management console.	Determines where the response is sent. The value of this parameter must exactly match one of the values listed for your application (including the http or https scheme and case). Your application must be able to parse the query string that Kiind adds to get the authorization code.
scope	PAYMENT or GIFT	PAYMENT scope allows use and lookup of Giftbit gift cards, and should be used if you are integrating Giftbit as a payment type in a point of sale or eCommerce checkout for your merchants. GIFT scope allows you to send gifts, and should be used if you are building gifting into your application.
state (optional)	Any string	Provides any state that might be useful to your application upon receipt of the response. The Kiind server roundtrips

		this parameter on the redirect_uri, so your application receives the same value it sent when we redirect the user back .

An example URL is shown below, with line breaks and spaces for readability only.

```
https://www.kiind.me/oauth/userlogin?client_id=SAMPLEAPP
&response_type=code
&scope=GIFT
&redirect_uri=https%3A%2F%2Fwww.yourapplication.com%2Fhandleredirect
&state=yourOptionallySuppliedState
```

Handling the response

The response will be sent by redirecting the user to the redirect_uri as specified in the request URL. If the user approves the access request, then the response contains an authorization code and the state parameter (if included in the request). If the user does not approve the request or there was a problem with the parameters you provided, the response contains an error message.

An error response:

```
https://www.yourapplication.com/handleredirect?error=access_denied&error_description=The
+user+canceled+the+authorization+request.
```

An authorization code response:

```
https://www.yourapplication.com/handleredirect?state=yourOptionallySuppliedState&code=u
b6Smbw7Px
```

Important: if your response endpoint renders an HTML page, any resources on that page will be able to see the authorization code in the URL. Carefully consider if you want to send authorization credentials to all resources on that page (especially third-party scripts such as social plugins and analytics). To avoid this issue, we recommend that the server first handle the request, then redirect to another URL that doesn't include the response parameters.

Step 2: Exchanging the authorization code for access and refresh tokens

After your web server receives the authorization code, it may exchange the authorization code for an access token and a refresh token via the token endpoint

token api endpoint
PRODUCTION https://api.kiind.me/oauth/token SANDBOX https://testbed.kiind.me/oauth/token

This request is an HTTPS POST, and includes the following parameters:

Field	Description
code	The authorization code returned from step 1.
client_id	The client ID for your application obtained from your Kiind integrations console TODO link
client_secret	The client secret obtained from your Kiind integrations console TODO link
redirect_uri	The same value for ‘scope’ you used in step 1.
scope	The same value for ‘scope’ you used in step 1.
grant_type	As defined in the OAuth 2.0 specification, this field must contain a value of authorization_code.

The actual request might look like the following (linebreaks in body for readability only):

endpoint https://api.kiind.me/oauth/token
method POST
headers: accept=application/json
contentType: application/x-www-form-urlencoded
body

```
grant_type=authorization_code
&code=ub6Smbw7Px
&client_id=SAMPLEAPP
&client_secret=141c13039c89425cb435a0e6fbbffd5b
&scope=GIFT
&redirect_uri=https%3A%2F%2Fwww.yourapplication.com%2Fhandleredirect
```

A successful response to this request contains the following fields:

Field	Description
access_token	The access token that can be used to make API calls for the authenticated user.
refresh_token	A token that may be used to obtain a new access token and new refresh token. Refresh tokens are valid until the user
expires_in	The remaining lifetime of the access_token, in milliseconds.
token_type	Identifies the type of token returned. At this time, this field will always have the value Bearer.

A successful response is returned as a JSON array.

Response

```
{
   "expires_in": 86400,
   "token_type": "bearer",
   "refresh_token":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJTSEEyNTYifQ==.R0FSNVNRZnJOeVEzTit3RjNGTjJqcEUxM
WNGeVNzMmVTSi9KZk02czJBY3E0UlVSL0tsL1NldG1BVThSaElOK1phZ0ZxMVZ1eVNoUXE
4Z2QvUTNzTlQ4VnVOSXpCRmRPMHljMnZoV0h3Y2ZTMFlpbFZtQ24wL2JhdUUxTFVFaUJre
Vh5U1B5cDdkcVB3cVU4ckU3ZTNRPT0=.yX3As569Jz7/RpWG3nNrYlSxfwy5IW8KlJwvKhG
gdPM=",
   "access_token":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJTSEEyNTYifQ==.TjZHZHVRS1ZsdThLeDRQOXZielc0dGtBL
1hQMWdwT0NZMFlTK0k5MjVXUGRZeStUYjBuTlRZQ0JXeTk4Q2YraW4vMU0vcjhndjJucHJ
FQWowaVd5aXpjVzQ4WW0vR282VVpERnhqN2ExY2xMQ1BtbDhvZEpva3BISmZNYU0zY09
OWGdOTE5TTTQzMHJKbDlnRHlyTWhnPT0=.6amPqJqC5w8PYVkTX+9heCvm5LP+VRAd/3
gGhYIuKQw="
}
```

Important: access and refresh tokens should be persisted securely.

Note: Other fields may be included in the response, and your application should not treat this as an error. The set shown above is the minimum set.

Step 3: calling the Kiind or Giftbit APIs

After your application obtains an access token, you can use the token to make calls to the Kiind or Giftbit API on behalf of the given user.

If you are accepting Giftbit giftcards, the documentation for the payment endpoints is here

If you are allowing your users to send gifts, the documentation for the gifting API exists here.

Using a refresh token

As indicated in the previous section, a refresh token is obtained at the same time as an access token. Access tokens are short lived for security purposes as they are sent over the wire frequently (their life-span from the current time is given by the expires_in in the response); refresh tokens are longer lived (6 months), and are used to get a new access token without requiring the user to authenticate again.

Important: it is the responsibility of your implementation to ensure you are using non-expired access tokens by storing the timestamp that corresponds to the current time plus the expires_in value, and using the refresh token once it the expiry has passed to retrieve a new access token with a new expiry (and a new refresh token).

refresh token api endpoint (same as token api endpoint)
PRODUCTION https://api.kiind.me/oauth/token SANDBOX https://testbed.kiind.me/oauth/token

This request is an HTTPS POST, and includes the following parameters:

Field	Description
refresh_token	The refresh token returned from the authorization code exchange.

client_id	The client ID obtained from the Developer portal.
client_secret	The client secret obtained from the Developer portal.
grant_type	As defined in the OAuth 2.0 specification, this field must contain a value of refresh_token.

The actual request might look like the following (linebreaks in body for readability only):

```
endpoint: https://api.kiind.me/oauth/token
method: POST
headers: accept=application/json
contentType: application/x-www-form-urlencoded
body
grant_type=refresh_token&client_id=SAMPLEAPP&client_secret=a8d6d310a4e747198b
e0c67002488c65&refresh_token=eyJ0eXAiOiJKV1QiLCJhbGciOiJTSEEyNTYifQ%3D%3D.
MjlBMUx4SzZNRDZEUjlJR0h3SWF5RUVhZXpTdW8wMVFpVktFcUZleGFSZUNSWFY5VVFU
MVhBUnhYakp5aEVFbHZMZ3FQMWhLYVluK3JCTm5vd0xBeWQwbDdsaWtIeFJHZFd4VDZ
WUWFIK3lWZ1NZNS91b3V4bmF1Y2N5UzhjWmYyU0YwZEM1R3lVWTZJRE93aFlmbm5BPT
0%3D.aa3aEcwnCNQHrhEjqr7U7SsXAa1S%2B0HFmJ6l5mC50FU%3D
```

As long as the user has not revoked the access granted to your application, the response includes a new access token, expiry, and a new refresh token, with identical response format as shown above in the authorization code for tokens section.

Token and Refresh Token Expiry Length

- Authorization codes are valid for 10 minutes. We suggest using them immediately to get the API_Token.
- API Tokens are valid for 24 hours.
- Refresh tokens are valid for 6 months, and are single use. When using a refresh token, ensure you store the new refresh token that is returned with the response (and replace/delete the one you just used) as you then never need to ask the connected user to reauthenticate.

Expiry Lengths in the Test Environment

To help with your testing of the token use and refresh process, we've made the expiries of the API and refresh tokens short - 5 minutes for the API token and 1 hour for refresh tokens

Offline access

Kiind does not differentiate between 'online' (when the user is present on your application) and 'offline' (when they are not) access. You may used obtained access tokens for user triggered or event driven Kiind API calls.

Help and Support

This stuff can be tricky! The Kiind Api team is always available to help out our integrators. For sandbox and non production questions while you are integrating: testbed@kiind.me - for production access requests or production account questions: apiaccess@kiind.me