2021-03-23

REST API v1

For use with:
https://openbadgefactory.com
https://factory.cancred.ca

Authentication

OAuth2 Client Credentials

OAuth2 bearer token authentication is available at Pro level subscription. This is the preferred method to access the API.

1. Get your API client_id and client_secret

Login to OBF as a user in admin role and got to Admin tools > API. Generate new client secret and give it an informative description that helps you identify it later. Note that the secret string is shown one time only, you must copy and store it at this point.

The client_id string identifies your organisation and it is used in most API route URLs.

2. Request an access token

`POST /v1/client/oauth2/token`

Body parameters:

Parameter	**Type**	**Description**
grant_type	String (required)	The grant type parameter must be set to "client_credentials"
client_id	String	Client credentials you obtained in step one.
client_secret	String	

Client credentials can be either included as POST body parameters or in HTTP Basic auth header.

Response JSON:

```
{
  "access_token":  "{your access token}",
  "token_type":    "bearer",
  "expires_in":    36000
}
```

You can use the same access token until it expires. Expiration time is short and you need to fetch a new access token after the old one becomes invalid.

The same client credentials can be used until your subscription to our service ends. You can view the list of generated keys in Admin tools > API and also revoke old credentials if necessary.

3. Test your access token

Include your newly created access token in request `Authorization` header:

```
Authorization: Bearer {your access token}
```

Run a test request:

```
GET /v1/ping/{$client_id}
```

Successful request will have return code 200 OK and the response body echoes back your client id.

Example, using curl:

```
$ curl -H "Authorization: Bearer $access_token" \
  "https://$hostname/v1/ping/$client_id"
```

More info:

https://www.oauth.com/oauth2-servers/access-tokens/client-credentials/

Client side certificates

X.509 client side certificates can be used to authenticate you and authorize API calls. The procedure for creating your certificate is as follows:

1. Get your API key
Login to OBF as a user in admin role and got to Admin tools > API key > Legacy. You need this token for generating your certificate signing request. The generated token will be valid for ten minutes. After that it cannot be used for certificate signing.

2. Get our public RSA key

```
GET /v1/client/OBF.rsa.pub
```

3. Decode your API key with our public key

in Perl:

```
my $key = Crypt::OpenSSL::RSA->new_public_key($pubkey);

$key->use_pkcs1_padding;

my $decrypted = $key->public_decrypt(decode_base64($apikey));

my $json = decode_json($decrypted);
```

in PHP:

```
$key = openssl_pkey_get_public($pubkey);

$decrypted = '';

openssl_public_decrypt(
  base64_decode($apikey), $decrypted, $key, OPENSSL_PKCS1_PADDING
);

$json = json_decode($decrypted);
```

4. Generate your Certificate Signing Request with your decoded API key

API key JSON object has the following structure:

```
{
  "id":      "...", // your client id
  "subject": "...", // your certificate subject line
  "ctime":   "...", // timestamp
  "nonce":   "..."  // security nonce
}
```

Id parameter is your client id in OBF. Save it for later use.

Use the subject parameter as the subject line in your CSR. (Command line OpenSSL used in this example.)

```
$ openssl req -new -nodes -batch -days 1095 -newkey rsa:2048 \
  -keyout /tmp/obf-test.key -subj '$PAYLOAD_SUBJECT' \
  > /tmp/obf-test.csr
```

5. Send your CSR for signing

`POST /v1/client/{$client_id}/sign_request`

```
{
    signature => "...",
    request   => "..."
}
```

Make a POST request with a JSON string as body content. `Signature` parameter is the base64-encoded api key you got from our admin panel (in step 1.) Just send it back as-is, without changes.

`Request` parameter is your certificate signing request file contents. Successful signing operation will have return code `200 OK` and the response body contains your new certificate.

6. Store your certificate and private key for future use

You need your certificate and private key with every API call. Store the files securely and grant appropriate file system permissions.

Also, remember to save your client id.

7. Test your keypair

Include your newly created keypair in your user agent (see examples below) and run a test request:

`GET /v1/ping/{$client_id}`

Successful request will have return code `200 OK` and the response body echoes back your client id.

Check your platform docs for the usage of client side certificates. Here are some examples.
Perl, with LWP:

```
my $ua = LWP::UserAgent->new;

$ua->ssl_opts(
    SSL_verify_mode => 'SSL_VERIFY_PEER',
    SSL_key_file    => '/path/to/your/private.key',
    SSL_cert_file   => '/path/to/your/certificate.pem'
);

my $res = $ua->get("https://$hostname/v1/ping/" . $client_id);

# Expected results:
#   $res->code    == 200
#   $res->content eq $client_id
```

PHP, with cURL:

```
$ch = curl_init();

$options = array(
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_SSL_VERIFYHOST => 2,
    CURLOPT_SSL_VERIFYPEER => true,
    CURLOPT_HEADER         => false,

    CURLOPT_URL     => "https://$hostname/v1/ping/" . $client_id,
    CURLOPT_SSLCERT => '/path/to/your/certificate.pem',
    CURLOPT_SSLKEY  => '/path/to/your/private.key',
);

curl_setopt_array($ch , $options);

$result = curl_exec($ch);
$info   = curl_getinfo($ch);
```

```
curl_close($ch);

/*
* Expected results:
*   $info['http_code'] == 200
*   $result == $client_id
*/
```

Formats and encodings

Unless otherwise noted, all input and output is JSON encoded. String encoding is always UTF-8.

Lists of objects are returned as Line Delimited JSON:

```
{"key1":"...","key2":"..." ... ,"keyX":"..."}\r\n
{"key1":"...","key2":"..." ... ,"keyX":"..."}\r\n
{"key1":"...","key2":"..." ... ,"keyX":"..."}\r\n
{"key1":"...","key2":"..." ... ,"keyX":"..."}\r\n
```

Badge Operations

Create new badge

`POST /v1/badge/{client_id}`

Input JSON object parameters:

Parameter	Type	Description
name	String (required)	Name of the badge.
description	String (required)	Description of the badge.
image	String	Base64 encoded PNG image data.
css	String	Criteria page styles.

criteria_html	String	Criteria page HTML.
email_subject	String	Email chunks for badge issuing message.
email_body	String	
email_link_text	String	
email_footer	String	
expires	Int	Unix timestamp for optional badge expiration date.
tags	Array	Array of strings representing badge tags.
draft	Boolean (required)	Badge status.
metadata	Object	Custom badge metadata. Optionally you can define your platform-specific metadata and store it with the badge. This field can be used to store any valid JSON object. The data is used internally only and it won't be present in issued badges.

Update badge

```
PUT /v1/badge/{client_id}/{badge_id}
```

Same input parameters as create. Doesn't affect already issued badges.

Delete badge

After deletion a badge is no longer available for issuing. Doesn't affect already issued badges.

Delete all badges:

```
DELETE /v1/badge/{client_id}
```

Delete single badge:

```
DELETE /v1/badge/{client_id}/{badge_id}
```

Get badges

Single badge by id:

```
GET /v1/badge/{client_id}/{badge_id}
```

All badges:

```
GET /v1/badge/{client_id}
```

Optional query parameters:

Parameter	Type	Description
draft	(0, 1)	Allows you to filter badges by status. If not present, all badges are returned.
category	String	Badges can belong in one or more categories. These are used for internal purposes only and are different from OBI badge tags. Multiple categories can be separated in query by pipe sign.
id	String	List of badge ids to fetch, separated by pipe sign.
query	String	Search badges by name or description.
meta:{key}	String	Filter badges by custom metadata.

Available badge categories in search:

```
GET /v1/badge/{client_id}/_/categorylist
```

Metadata search example:

Given badge metadata field:

```
{"foo":123, "bar":456, "baz":"quux"}
```

Matching query:

```
?meta:foo=123&meta:baz=quux
```

No match:

```
?meta:foo=124
```

Issue badge

When you issue a badge your recipients will get an email message containing an url where they can accept the badge and add it to their passport or backpack.

```
POST /v1/badge/{client_id}/{badge_id}
```

Input JSON object parameters:

Parameter	**Type**	**Description**
recipient	Array (required)	List of recipient email addresses.
expires	Int	Unix timestamp for optional badge expiration.
issued_on	Int	Unix timestamp, current time is used by default.
email_subject	String	Email chunks for badge issuing message. Required, unless the message has been stored with the badge.
email_body	String	
email_link_text	String	
email_footer	String	
badge_override	Object	Overrides badge data for this particular issuing event. Allowed keys: ● name ● description

		• criteria • tags Values provided here will replace the content of the badge. In addition, key criteria_add can be used to append content to the original criteria text.
log_entry	Object	A log entry object of unspecified format can be saved along the issuing event.
api_consumer_id	String	Parameter identifying this client.
send_email	(0, 1)	Send/don't send email messages to badge recipients. If value is 0, you must provide a way for users to get their badges. Default value 1.

Return code: `201 Created`

`Location` response header field contains the url of this issuing event.

Badge revocation

After you revoke a badge, GET requests to it's assertion url will return `410 Gone` response. This tells displayers that the badge is no longer valid.

Badges are revoked with an issuing event id and one or more recipient email addresses. If a recipient has been issued the same badge multiple times, only the one identified by event id is revoked.

See Report section of this document for more information on issuing event operations.

Revoke a badge:

```
DELETE /v1/event/{client_id}/{event_id}/?email=foo@example.com
```

```
DELETE
/v1/event/{client_id}/{event_id}/?email=foo@example.com|bar@example.
com|quux@example.com
```

Return code: `204 No Content`

Get revoked badges:

```
GET /v1/event/{client_id}/{event_id}/revoked
```

Sample output:

```
{
    "revoked": {
        /* recipient email and revocation timestamp */
        "foo@example.com": 1434971021,
        "bar@example.com": 1434971020,
        ...
    }
}
```

Issuer Operations

Get own data

```
GET /v1/client/{client_id}
```

Update own data

```
PUT /v1/client/{client_id}
```

Input JSON object parameters:

Parameter	Type	Description
url	String (required)	Issuing organization's web address.
description	String (required)	Description of the issuer.
email	String (required)	Canonical email address of the issuer.
image	String	Base64 encoded image representing the issuer, a logo.

Badge Applications

Get earnable badges

List all:

`GET /v1/earnablebadge/{client_id}`

Query parameters:

Parameter	Type	Description
badge_id	String	Find by badge id.
appoval_method	(“review”, “instant”, “secret”, “peer”)	Find by approval method.
visible	(0, 1)	Filter by visibility status.
client_alias	String	Filter by client alias id.

Single earnable badge by id:

`GET /v1/earnablebadge/{client_id}/{earnable_id}`

Get applications

List all applications to a badge:
`GET /v1/earnablebadge/{client_id}/{earnable_id}/application`

Query parameters:

Parameter	Type	Description
status	(“approved”, “pending”, “rejected”)	Filter by application status.

Single application by id (contains application form data):

`GET /v1/earnablebadge/{client_id}/{earnable_id}/application/{ap_id}`

Process applications

Approve or reject:

`PUT /v1/earnablebadge/{client_id}/{earnable_id}/application/{ap_id}`

Input JSON object parameters:

Parameter	Type	Description
result	(“approve”, “reject”)	Result of assessment.
reviewer	String	Reviewer’s identification string.
expires	Int	Approval parameters, see badge issuing section for details.
issued_on	Int	
email_subject	String	
email_body	String	
email_link_text	String	
email_footer	String	
log_entry	Object	
api_consumer_id	String	

If an application is rejected, you can optionally send a rejection message using `email_subject` and `email_body` parameters. Other issuing parameters are not used in this case.

Reports

Get issuing events

Single event by id:
`GET /v1/event/{client_id}/{event_id}`

Search for events:
`GET /v1/event/{client_id}`

Query parameters:

Parameter	Type	Description
api_consumer_id	String	Filter by issuer plugin id.
badge_id	String	Filter by badge id.
email	String	Filter by recipient email address.
begin	Int	Filter by issuing event date range. Unix timestamps.
end	Int	
order_by	(“asc”, “desc”)	Order results by date, ascending or descending.
limit	Int	Maximum number or results to return, upper limit 1000.
offset	Int	Skip a number of events. Used with limit parameter when paginating results.
count_only	(0, 1)	If true, the query returns only the number of results found with given parameters.

Get badges and assertions

Download issued badges in different formats:
`GET /v1/event/{client_id}/{event_id}/assertion`

Query parameters:

Parameter	Type	Description
email	String	Filter by recipient email address (optional, use | to separate multiple addresses).

Sample output, multilingual badge:

```
// PDF download links for a multilingual badge
{
    "id": "assertion_id_1",
    "image": "{image url 1 }",
    "json": "{assertion url 1}",
    "pdf": {
      "en": "{PDF file url 1 - en}",
      "fi": "{PDF file url 1 - fi}",
      "fr": "{PDF file url 1 - fr}",
      "sv": "{PDF file url 1 - sv}",
    },
    "recipient": "user1@example.com",
    "status": "unknown"
}
...
{
    "id": "assertion_id_X",
    "image": "{image url X}",
    "json": "{assertion url X}",
    "pdf": {
      "en": "{PDF file url X - en}",
      "fi": "{PDF file url X - fi}",
      "fr": "{PDF file url X - fr}",
      "sv": "{PDF file url X - sv}",
    },
    "recipient": "userX@example.com",
    "status": "accepted"
}
```

Sample output, single-language badge:

```
// PDF download links for a single-language badge
{
    "id": "assertion_id_3",
    "image": "{image url 3}",
    "json": "{assertion url 3}",
    "pdf": {
      "default": "{PDF file url 3}",
    },
    "recipient": "user3@example.com",
    "status": "accepted"
}
...
{
    "id": "..."
}
```

JSON and image formats can be downloaded without an API key. Image and PDF files contain baked Open Badge metadata.

PDF downloads are rate-limited to one download/second. PDF format is available on special subscription level. Please contact our sales team for more information.

Return codes

200 OK
Successful GET responses.

201 Created
Resource created successfully. Check `Location` header field for new id. Response body will be empty.

204 No Content
Successful PUT and DELETE responses.

Errors

400 Bad Request
Invalid parameter(s), e.g. missing or of wrong type.

403 Forbidden
Client is not authorized to access resource.

404 Not Found
Requested resource cannot be found.

405 Method Not Allowed
HTTP method is not supported/recognized

411 Length Required
POST or PUT request length missing.

413 Request Entity Too Large
POST and PUT requests are limited to maximum size of 67108864 bytes.

429 Too Many Requests
Too many API calls per second. Check Retry-After header for cooldown time (in seconds).

495 Cert Error
Invalid client side certificate.

496 No Cert
Certificate missing.

500 Internal Server Error
Unexpected fatal error, a bug.

503 Service Unavailable
The service is temporarily closed for maintenance or other reasons. Client should retry the request after a short period.

Changes

2021-03-23

- Issue badge, added `send_email` parameter
- Added received status to GET /v1/event/{client_id}/{event_id}/assertion