API AUTHORIZATION

INTRODUCTION

The TicketNetwork APIs use the OAuth 2.0 (bearer token) authentication method for API and resource authorization. Tokens need to be regenerated approximately every 60 minutes and provided with each API call to authenticate and authorize access to the APIs.

CONTENTS

API CLIENT CREDENTIALS

Each individual application has a unique set of client credentials that identify that application, and which are used to generate access tokens.

An application's credentials uniquely identify it to TicketNetwork's system. Never expose your application's consumer secret or the base64-encoded value to anyone outside of TicketNetwork support. This would allow them to generate access tokens, masquerade as your application, and do anything that your application could do. Treat this information as you would a username and password.

Each application has a separate set of credentials for the Sandbox and Production environment. A token generated by the application's Sandbox credentials cannot be used for authorization against Production, and vice-versa. Your Sandbox and Production application credentials for each application in your integration were provided to you during the onboarding process. If you have any questions about your credentials, please contact TicketNetwork Support.

The following information is used to obtain an OAuth2 access token.

- **Consumer Key**: The public key of the application, used to uniquely identify the application in the environment. This value does not need to be kept secret. Consumer key will be different between Sandbox and Production for the same application.
- **Consumer Secret**: The private key of the application for the environment (again, Sandbox and Producation will differ). Used along with the consumer key to generate access tokens for the application. Treat this information as you would a password. Consumer secret will be different between Sandbox and Production for the same application.
- **Base64-encoded Value**: The base64-encoded combination of consumer key and secret, used as the Basic header to generate and revoke application access tokens. Create this value programmatically by using a base64 encoding function `base64encode(consumerKey + ":" + consumerSecret)`. Treat this information as you would a password.

OBTAINING AN ACCESS TOKEN

NOTE: Generating a new access token with the same scopes as an existing token will invalidate the previous token. One JWT access token can be used multiple times within its lifetime, and it is best practice to re-use the same token for as long as possible after generating it.

To generate an access token, integrate with the OAuth Token API. Use the /oauth2/token resource to obtain a new token.

Combine your application's consumer key and secret in the format `consumer-key:consumer-secret`, then base64 encode the resulting string. This will be used as a Basic auth token in the Authorization header of the HTTP request. Use the Sandbox client credentials when generating a Sandbox application token, and the Production credentials when generating a Production application token.

Submit a form POST request to the /oauth2/token resource to generate a new token and invalidate the old token. Include any scopes needed by that token in the request.

Access Token Generation, HTTP Request Information	
Method	POST
URL	`https://key-manager.tn-apis.com/oauth2/token`
HTTP Headers	
Content-Type	`application/x-www-form-urlencoded`
Authorization	`Basic <AppCredentialsBase64Value>`
HTTP Body	
grant_type	`client_credentials`
scope	`<the list of scopes separated by a space, plus (optionally) a device scope>`

When successful, the response will return with an HTTP status code of 200 (OK), and the body will contain a JSON object with the new access token as well as the number of seconds remaining before the token expires.

Token HTTP Response Information	
JSON BODY	
access_token	`<the newly generated token>`
scope	`<space-separated list of scopes granted by the token>`
token_type	`Bearer`
expires_in	`<the number of seconds that the token will be valid for>`

EXAMPLE REQUEST

```
POST https://key-manager.tn-apis.com/oauth2/token HTTP/1.1
Authorization: Basic xxxxxxxxxxxxxxxxxxx
Host: key-manager.tn-apis.com
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate

grant_type=client_credentials&scope=api_resource_scope_1 api_resource_scope_2
```

EXAMPLE RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "access_token": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-xxxx-xxxxxxxxxxx-xxxxxxxxxxxxxxxxxxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-xxxxxxx.xxxxxxxxxxxxxxxxxxxxxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxx-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-xxxxx-xxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
    "scope": " api_resource_scope_1 api_resource_scope_2",
    "token_type": "Bearer",
    "expires_in": 3600
}
```

WHEN TO GENERATE A TOKEN

Once generated, a JWT access token will be valid for about one hour. Generate another new token a couple of minutes before the current token expires. To calculate the expiration datetime of a token, add the expires_in value (in seconds) to the current time after receiving the new token.

It's also possible that, due to security reasons, an existing token might sometimes be revoked by the system, or by TicketNetwork staff. This will be a rare occurrence, but to ensure uninterrupted service, integrators should account for this possibility.

If an application makes an API request with a token that should be valid, and it receives a 401 (Unauthorized) response with a fault code of 900901, then the application should generate a new token and retry the API request.

EXAMPLE API RESPONSE WHEN THE TOKEN IS EXPIRED OR REVOKED

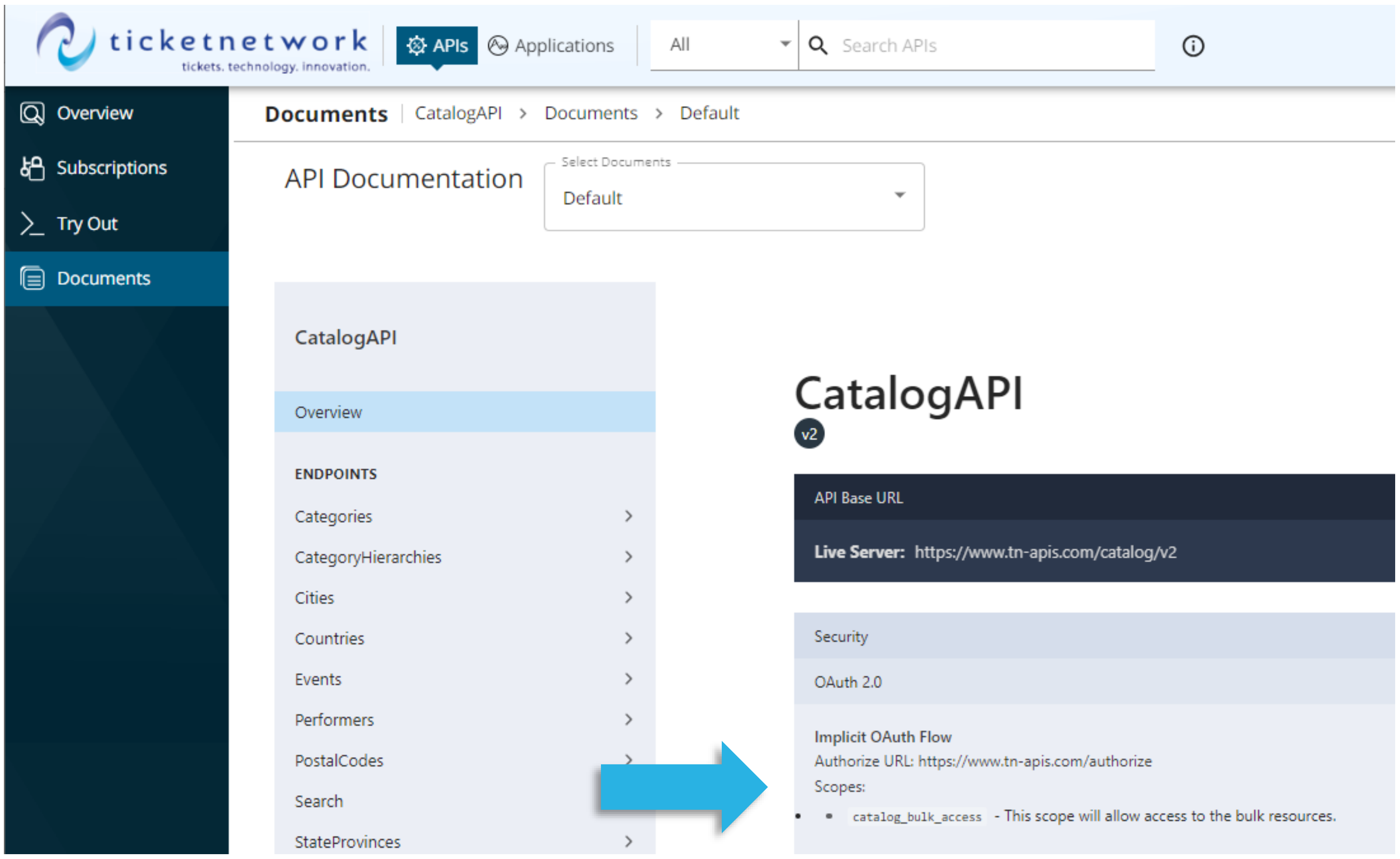
```
HTTP/1.1 401 Unauthorized
Content-Type: application/json; charset=UTF-8

{
    "fault": {
        "code": 900901,
        "message": "Invalid Credentials",
        "description": "Access failure for API: /api/v1, version: v1 status: (900901) - 
Invalid Credentials. Make sure you have provided the correct security credentials"
    }
}
```

WHAT SHOULD I USE AS MY SCOPES?

Scopes allow access to additional restricted resources on an API. The actual scopes vary from API to API, and not all scopes are allowed for all integrators.

Which scopes your application needs will depend on the APIs you are using, as well as your integration. If your integration needs to call a resource that is restricted via a scope, then make sure to include that scope in the request when generating your access token. Otherwise, requests to the individual resource that requires the scope will be denied.

You can browse all the available scopes for an API by navigating to the Dev Portal, then going to that API's Documents page. Scopes are listed on the Default Document Overview, under OAuth 2.0. When requesting scopes for a token, only the scopes that are authorized for your individual integration will be included in the generated token.



API Base URL

Live Server: https://www.tn-apis.com/catalog/v2

Security

OAuth 2.0

Implicit OAuth Flow
Authorize URL: https://www.tn-apis.com/authorize
Scopes:

- `catalog_bulk_access` - This scope will allow access to the bulk resources.

TOKENS IN A DISTRIBUTED ENVIRONMENT

As previously stated, generating an access token will automatically revoke the existing token *with the same scopes* for that application. Tokens are maintained per-application, so generating a token for, say, MyApp1 would never revoke a token that has been issued for MyApp2 even if those tokens share the same scopes. However, the previous token issued to MyApp1 *would* be revoked.

It is possible to create a distributed system wherein the same JWT token is synced and used across multiple separate processes. However, for integrators who prefer to have multiple active tokens at once, you can add a `device_<unique-device-id>` scope to the token request and generate one token per device.

For example, assuming a deployment setup with the same application running on Instance A and Instance B:

Instance A would generate its token by including the following scope in the /oauth2/token request: `device_instance-a`

Instance B would generate its token by including the following scope: `device_instance-b`

This would result in two overall valid tokens – one token for device “instance-a”, and one token for device “instance-b”.

EXAMPLE REQUEST WITH A DEVICE SCOPE

```
GET https://key-manager.tn-apis.com/oauth2/token HTTP/1.1
Authorization: Basic xxxxxxxxxxxxxxxxxxx
Host: key-manager.tn-apis.com
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate

grant_type=client_credentials&scope=api_resource_scope_1 api_resource_scope_2
device_instance-a
```

AUTHENTICATING AN API REQUEST

ADDING AN ACCESS TOKEN TO THE API REQUEST

Once obtained, include the JWT access token in the Authorization HTTP header as a Bearer token. Make sure the API request includes an ‘Accept’ header with the content type for the API.

EXAMPLE REQUEST

```
GET https://sandbox.tn-apis.com/api/v1/api_resource HTTP/1.1
Authorization: Bearer xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-xxxx-xxxxxxxxxxx-xxxxxxxxxxxxxxxxxxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-xxxxxxx.xxxxxxxxxxxxxxxxxxxxxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxx-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-xxxxx-xxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Host: sandbox.tn-apis.com
Connection: keep-alive
Accept: application/json
Accept-Encoding: gzip, deflate
```

ADDING A CONSUMER KEY TO THE API REQUEST

Some API calls authenticate using just the application’s consumer key instead of using the JWT access token directly. Use consumer key-based authentication for JavaScript client-side Maps API integrations by including your application’s consumerKey parameter right in the query string of the API call.

Although client-side calls to Maps API use consumer key-based authentication, integrators are still required to periodically generate JWT access tokens using the steps outlined in the *Obtaining an Access Token* section. Failing to properly generate an access token will result in client-side requests to Maps API being denied.

EXAMPLE REQUEST

```
GET https://sandbox.tn-
apis.com/maps/v3/MapAndLayout?consumerKey=xxxxxxxxxxxxxxx&websiteConfigId=123&eventId=203
518 HTTP/1.1
```

REVOKING AN ACCESS TOKEN

If you ever want to manually revoke an access token, you can do so by using the /oauth2/revoke endpoint.

As with token generation, combine your application's consumer key and secret in the format `consumer-key:consumer-secret`, then base64 encode the resulting string. This will be used as a Basic auth token in the Authorization header of the HTTP request. Use the Sandbox client credentials when generating a Sandbox application token, and the Production credentials when generating a Production application token.

Submit a form POST request to the /oauth2/revoke resource to revoke the token.

Access Token Revocation, HTTP Request Information	
Method	POST
URL	`https://key-manager.tn-apis.com/oauth2/revoke`
HTTP Headers	
Content-Type	`application/x-www-form-urlencoded`
Authorization	`Basic <AppCredentialsBase64Value>`
HTTP Body	
token	`<the token to revoke>`

When successful, the response will return with an HTTP status code of 200 (OK) a blank response body.

Revoke HTTP Response Information	
HTTP Headers	
AuthorizedUser	`<the username of the account that the revoked application token belonged to>`
RevokedAccessToken	`<the OAuth2 access token that was revoked>`
RevokedRefreshToken	`<the OAuth2 refresh token that got revoked along with the access token>`

EXAMPLE REQUEST

```
POST https://key-manager.tn-apis.com/oauth2/revoke HTTP/1.1
Authorization: Basic xxxxxxxxxxxxxxxxxxx
Host: key-manager.tn-apis.com
Accept: application/json
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate

token=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-
xxxx-xxxxxxxxxxx-xxxxxxxxxxxxxxxxxxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-xxxxxxx.xxxxxxxxxxxxxxxxxxxxxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxx-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-xxxxx-xxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

EXAMPLE RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json
AuthorizedUser: xxxxx@carbon.super
RevokedAccessToken: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-xxxx-xxxxxxxxxxx-xxxxxxxxxxxxxxxxxxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-xxxxxxx.xxxxxxxxxxxxxxxxxxxxxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxx-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-xxxxx-xxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
RevokedRefreshToken: xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx
```

SECURITY

Do not share your access tokens, consumer secret, or your application's base64-encoded credentials with anyone outside of TicketNetwork support. Doing so will allow others to make API calls as though they were you. This not only gives them access to your information and lets them perform actions through your account, it will also count against your API throttle limits.

If your account data has been compromised, please contact your account representative immediately, and they will work with you to rectify the problem and review the activity on the account.