Introduction to API Štíteknabalík.cz

Verze: **1.73**; 2025-11-05

Štíteknabalík.cz API is based on REST architecture. In order to be able to use it you need to fulfill following requirements.

1. Access only via secured **HTTPS** protocol.
2. Get authentication to access via https://id.app.stiteknabalik.cz/oauth-registration.
3. Authenticate using **HTTP Bearer token authentication**.
4. Send data encoded as **UTF-8**.

API environment

	Production environment PROD
Rest API 1]	https://rest.stiteknabalik.cz
Registration of OAuth client 2]	https://id.app.stiteknabalik.cz/oauth-registration
Registration of account	https://id.app.stiteknabalik.cz
Client app	https://id.app.stiteknabalik.cz/login

1] Rest API versions /v2, /v3, /v4.

2] OAuth client to connect to API requires company account creation.

Authentication

Štíteknabalík.cz supports two kids of authentication. **Basic** using generated api key and **Bearer** using OAuth 2.0., which supports scopes (access rights for particular methods)

All requests except for root address rest.stiteknabalik.cz require HTTP authentication. To access using **Basic auth** you need to get `token` from support team and to access using **OAuth** you need to get `access_token`.

Considering the nature of this authentication it is **NECESSARY to always use HTTPS protocol**. Single API call via HTTP compromises authentication tokens. More in section Authentication

HTTP codes

First verification on your side should be http status code verification.. All codes are according to HTTP specification.
(More about HTTP here https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html).

Possible HTTP codes:

HTTP code	Meaning	Details
200	**OK**	Request succesfully processed
201	**Created**	Request processed and resource succesfully created
304	**Not Modified**	Response identical to cached resource - empty body returned
400	**Bad Request**	Bad request received - invalid data input provided
401	**Unauthorized**	Authentication failed, auth header missing or invalid
404	**Not Found**	Requested resource (URL) not found
405	**Method Not Allowed**	Invalid http method used, response contains vlaid methods
406	**Not Acceptable**	Invalid Accept used
410	**Gone**	Old version of API used
412	**Precondition Failed**	Attempt to modify resource that was already updated
414	**Request-URI Too Large**	Too long URI used
415	**Unsupported Media Type**	Invalid Content-type header received
422	**Unprocessable Entity**	Request validation failed, for exmaple invalid package weight
426	**Upgrade Required**	Used unsecured HTTP protocol
500	**Internal Server Error**	Server side error
503	**Service Unavailable**	Service temporarily unavailable(service update in progress)

Supported communication formats

To make it easy to connect to our API we support several data formats of communication.

- use MIME type `Content-Type` header in your requests, to specify what data format you send us.
- Use MIME type `Accept` header in your requests, to specify what data format you expect in response. Supported formats in table bellow.
- Use `Accept-language` header in your requests, to localize messages in responses. We support `en` (as default) and czech `cs`.

Supported formats (for `Content-Type:` and `Accept:`):

JSON	Recommended	Use MIME type `application/json` in header.
XML	Support will be terminated	Use MIME type `application/xml` in header
form-urlencoded	Support will be terminated	Use MIME type `application/x-www-form-urlencoded` in header

Can I skip Accept header?

If header is missng we use generic `*/*`, as default format we use JSON. In case of invalid or unsupported header Accept the error message is in JSON.

What is the root element of XML?

If you decide to use XML format for reponse the root tag is always `<root>`.

How do I know I use wrong format?

If we accept invalid format, HTTP status 400 is returned.

What date format is used?

Unless specifie otherwise, all date formats are ISO 8601, eg. `2014-12-31T18:25:50+02:00`.
Více o ISO 8601 zde https://en.wikipedia.org/wiki/ISO_8601.

What decimals format is used?

You can use , or . equally. Separator of thousands should not be used, eg. `12500.00`, `330,50`.

> **Exceptions**
>
> If there is an error on our side the response can be in different format than requested in Accept header. These errors can occur on server side errors or during maintenance thus we recommend to check status code before parsing the reponse. An example can be 414 code. If the response code is 5xx, it is possible that response is not in expected format.

Response structure

All responses keep same strucutre. Particular repsonses depend on context but generally speaking they keep following structure.

Example of response structure

Successful response:

```
root
--- code
--- status
--- message
--- data
```

Error response:

```
root
--- code
--- status
--- message
--- errors
--- --- [0]
--- --- --- message
--- --- --- field
--- --- --- value
--- --- [1]
--- --- --- message
--- --- --- field
--- --- --- value
```

Example structure of error responses

JSON

```
{
"code": 422,
"status": "error",
"message": "Validation failed",
"errors":
    [
     {
        "message": "This value should be of type
float.",
        "field": "[0].packages[0].weight",
        "value": "3 kg"
     },
     {
        "message": "Invalid currency format,
expected ISO 4217",
        "field": "[0].valueCurrency",
        "value": "€"
     }
    ]
}
```

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <code>422</code>
    <status>error</status>
    <message>Validation failed</message>
    <errors>
        <message>This value should be of type float.
</message>
        <field>[0].packages[0].weight</field>
        <value>3 kg</value>
    </errors>
    <errors>
        <message>Invalid currency format, expected ISO
4217</message>
        <field>[0].valueCurrency</field>
        <value>€</value>
    </errors>
</root>
```

Description of items:

- `code` general error code identical to http code
- `status` response status, possible values: success, error

- `message` general description of error (in english)
- `data` response data, only for successsul repsonses, optional and structure depends on context
- `errors` array of errors for error repsonses, optional
- `errors.message` english desription of error
- `errors.field` path to error field
- `errors.value` invalid field value

> **Syntax**
>
> all keys use **camelCase** syntax.

> **More information**
>
> Detailed description of error codes and structure of succesfull response can be found on page of another our service Balíkonoš next to particular service methods.
> These services and resources are identical with Štíteknabalík.cz except for different endpoint here (https://rest.stiteknabalik.cz/api/v3/).

API versioning

The version of API used is part of URL. Right now there is API v4, thus URL of all resources begin with `https://rest.stiteknabalik.cz/api/v4/`. Any API version updates will be announced in advance. Older versions of API will be kept for some time.

Particular API versions:

- **v1** - initial API version, not supported anymore
- **v2** - to be deprecated soon - compatible with https://balikonos.cz/api/v2
- **v3** - actual version, compatible with https://balikonos.cz/api/v3
- **v4** - actual version, newest version updated regularly.

Testing and debugging

During development you will need to verify the behaviour of your application and what responses can you expect from API.

Creation of account

You can register to test nevironment using URL under this paragraph. It is a separate environment from PROD that uses different DB and doesn't call carrier APIs.

In Štíteknabalík.cz app you need to setup your colleciton places. For testing of particular carrier choose settings and inserta any testing credentials.

For testing of creation and closing of packages you need to **enable collection place** directly in carriers settings page.
Without this settings all packages will be marked as address to address, which is undesirable in cases when you want to test deliveries from collection place.

Links to create testing account can be found in section Environments.

CURL

Simplest way to test is using CURL Example usage:

```
curl -H "Accept: application/xml" -H "Authorization: Bearer bacffecfc1bd349b85e51b37a542aed57f457a8e"
https://rest.stiteknabalik.cz
```

Other tools

There is a vast variety of tools to test REST APIs, for exmaple Chrome extension Postman

Vlastnosti a funkce

Data compression

If you add `Accept-encoding: gzip` header to your request, responses will be compresseed using **gzip**. This method will save around **80 % of transfered data!** For JSON format approximately 65 % of data.

Compressed data will be return along with `Content-encoding: gzip` header. You need to decompress the data on your side. For example PHP You can also send use the requests using this method. You need to use `Content-encoding: gzip` header.

You can also configure your server to do the compression automatically - for example apache

> **Recommendation**
>
> Because amount of transferred data can be really big we **strongly recommend to always use compression**.

HTTP Cache

All `GET` methods return `ETag` header, with actual repsonse hash. This value should be stored along with returned data. In case of identical query with `If-None-Match` header no body is returned as content remains the same. Unless content changed, HTTP response will be 304 with empty body – as the content has not changed and it is unnecessary to transfer any data. This is useful in case of large repsonses(like PDF labels). Behavioud can be checked in browsera https://rest.stiteknabalik.cz

Další funkce

Overriding POST method

If you rapplicatoin cannot use PUT or DELETE HTTP requests, it is possible to use POST. If you set up `X-HTTP-Method-Override` headerto some of supported mathods (PUT či DELETE) using POST request, it will perform same action as if PUT or DELETE was used.

Only POST can be overridden, not GET. Method GET is safe and cannot change resource state.

Output format

Standard JSON and XML responses are pretty formated for better readability.This behaviour can be disabled using query parameter `prettyPrint` set to `false`. See example of root resource https://rest.stiteknabalik.cz?prettyPrint=false. This can also reduce the amount of transferred data

Unsupported functions

JSONP

Support for JSONP is not possible due to header authorization.

CORS

Support for Cross-origin resource sharing is not implemented nor planned to be.

Autentizace

Štíteknabalík.cz API supports simple **Basic** authentication and more advancerd **Bearer** OAuth 2.0.
In order to securely access API exposed on rest.stiteknabalik.cz **HTTPS** protocol, **HTTP** connections will be rejected.

Authentication method comparison

	Basic	Bearer (OAuth 2.0)
Support for particular requests	NO	YES
Token expiration	NO	YES, 60 minutes
Integration complexity	Low	High
Activation method	Generation of basic token in settings of Štíteknabalík.cz app (section API) [1]	Generation of OAuth client via form here

1] Basic token is immediatelly active after creation. It is immediatelly paired with you Štíteknabalík.cz registration and no other steps are needed. Basic token can be regenerated anytime.

2] OAuth client is not paired with your registration in Štíteknabalík.cz. Pairing is done afterwards - see "request for auth code" bellow.

Basic autentication

To obtain Basic API token please contact foxdlei support team.

They will generate token for you, that is required in the header of each request over HTTPS protocol. Token has limited lifetime and we recommend to ask for regeneration once in a while.

Basic token should be used in request header. Remember to keep one space between Basic and API token or the header iwll be considered invalid.

Example header:

```
Authentication: Basic 684v98fd84vfd9845df55616d5f7d10d54ce9798ccd391735387ea2f6d9c64ce7d5
Content-Type: application/json
```

> **Info**
> Possibility to manually generate Basic API tokenis currently under development.

OAuth 2.0

OAuth 2.0 protocol to authorize API clients was created according to specification RFC 6749.

This documentation describes steps to integrate and use OAUTH 2.0, so that working with the protocol is understandable and clear.
DEtails of particular parts of protocol are not listed but can be found in aforementioned specification or in this article.

Steps to activate OAuth client

1. **Registration of OAuth client** - register new OATUH 2.0 client for your API on our web
2. **Allowing authorization (request for auth code)** - Allow access to your API in your Štíteknabalík.cz account and save `authorization code`

3. **Request for access token** - Request `access_token` using your authorization code
4. **Access to resources** - Start using API according to API Štíteknabalík.cz specification
5. **Access to resources - refresh access token** Obtain new `access_tokenu`. Can be used only when you already have a `refresh_token`.

Step 1. - Register OAuth client

Make a registratin of new OAuth 2.0 client entering basic input data about your application, which will be using API access to your Štíteknabalík.cz data:
PROD https://id.app.stiteknabalik.cz/oauth-registration

Most important item in this form is **redirect URi** (described as `redirect_uri` in specification). It defines **endpoint**, To which our application will redirect all auth requests. **This address should use https protocol** as it uses sensitive data!

Password is your authentication secret, described in specification as `client_secret`. This secret is used to request `access_token`, comparably to password in Basic authentication. As a **username** in this OAuth authenticaiton you can use generated **identifier**, that you iwll obtain upon finished OAUTH client registration. In specification desribed as `client_id`.

A copy of registraion data will be sent to your email. It will contain aforementioned **identifier** and **password**, that you have set.

> **Warning**
> This implementation doesn't support login using query parameters or in request body. Find bellow sample request with this authorization. At the same time we do not support public clients usually implemented in javascript.

Step 2. - Enabling authorization(request for auth code)

You already have created an OAuth client and now you have to enable access to your Štíteknabalík.cz account. After approval you will save `authorization_code` to be used in next step.

This is done on authorization endpoint having following address:
`https://app.stiteknabalik.cz/oauth/authorize/` for PROD.

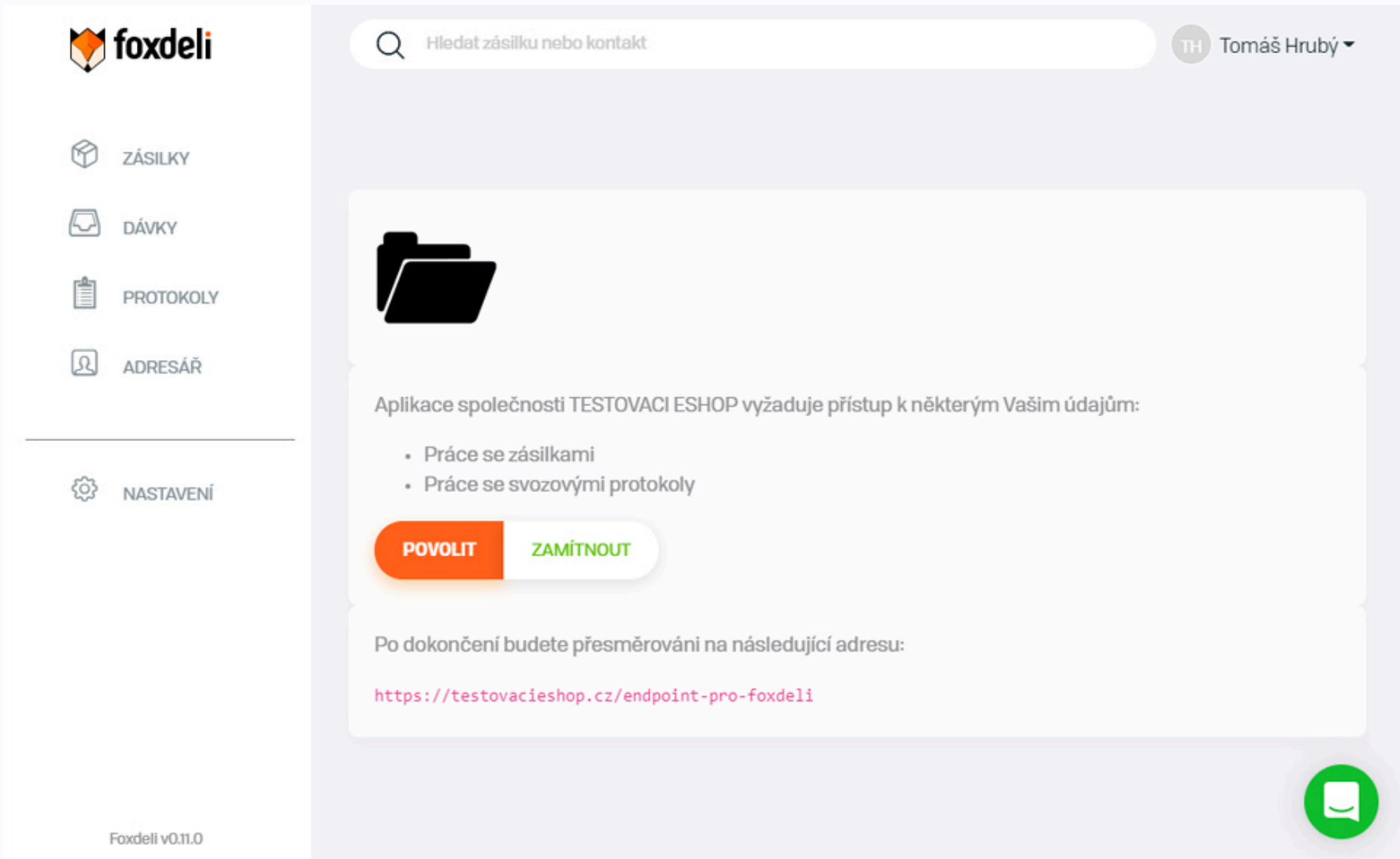
This urls **query stringu** contains following required parameters: `response_type`, `client_id` and also `scope` and `state`.
Scope defines what methods can OAuth client call. Scope of particular resources are defined in method documentation http://localhost:8880/en/methods#methods-endpoints (see OAuth Scope).
In case of multiple scopes join them using plus(+) character.
It is required that your client verifies accordance of **state** query parameter with reply from the server sent with random value of this param. This is used to prevent **CSRF attacks**.

In case you will also add `redirect_uri` parameter, it is required to be in strict accordance with address that you entered during registration of your OAuth client.

After succesful request user is displayd a form to allow or disallow your applications access to Štíteknabalík.cz data. Along with that name of your company, logo, web address, scope and redirect uri is displayed.
After redirection of URL you can obtain **authorization_code**. it will be passed via GET as a query parameter `code`.

> **WEBVIEW**
> To request for **authorization_code** you must be signed in Štíteknabalík.cz app account. It is necessary that system you use supports HTML page rendering and supports sessions. Because it is **GET** method, request can be done via regular browser. Following redirect can be directly to your API endpoint, where **authorization_code** will be catched and can be used by your system to authorize.

You allowing access, your logged user and manual confirmation of your ccount is part of securing the API access.



Page with allow button

In case of pressing allow button you will be redirected to `redirect_uri` where query string will contain `authorization_code` valid for 90 seconds. This code is always 40 characters of lowercase letters and numbers.

> **Warning**
> ŽRequest for `authorization_code` should only be done once or in case of `scope` change of given client. The goal is to get `authorization_code`, that can be used later on to obtain `access_token` and `refresh_token`.
> Once your `access_token` expires you can renew it using `refresh_token`. **Authorization via "webview" is not necessary anymore**.
>
> If you have multiple accouns in Štíteknabalík.cz application plase make sure you use the correct account to allow OAuth client.

Example of authorization request

Detailed request description can be found in specification.

Endpoint otevřít

```
GET: https://app.stiteknabalik.cz/oauth/authorize/?
response_type=code&client_id=zjhygknkfk&scope=deliveries+collection-
protocols&redirect_uri=https://mujeshopik.cz/stiteknabalik-endpoint/&state=csjkhd5b1
```

Example of succesful auth request (redirect)

Detailed description can be found in specification.

Endpoint otevřít

```
GET: https://mujeshopik.cz/stiteknabalik-endpoint/?code=87f90ba9fe97e3b43f11c37eea1e1b475dbd59b9&state=csjkhd5b1
```

Example of unsuccessful auth request

Detailed description can be found in specificaiton.

Endpoint otevřít

```
GET: https://mujeshopik.cz/stiteknabalik-endpoint/?
error=access_denied&error_description=The+user+denied+access+to+your+application&state=csjkhd5b1
```

Step 3. - request access token

In this step you will exchange `authorization_code` for new `access_token` and `refresh_token`.

Using `access_tokenem` you can access API for (**60 minutes**).

Afte3r expriation of `access_tokenu` you request new using `refresh_tokenu`. Refresh token has unlimited expiration.

> **TIP**
> Due to short expiration time of access token (and necessary renewal using refresh token) it is recommended to store refresh token in DB (if you create application for multiple account registered in Štíteknabalík.cz) or to appliation configuration (if you create an app just for you).
> Access token can be stored in session until it expires in order to decrease overhead necessary to access this API resource.
> It is not necessary to store access token in DB.
> It is strongly discouraged to request new access token for every request!

Example of HTTP request for access token (REQUEST)

Detailed description can be found in specification.

Endpoint

```
POST: https://rest.stiteknabalik.cz/oauth/token/
```

Ukázka hlavičky

```
Authorization: empoeWdrbmtmazphYmNkMTIzNA==
Content-Type: application/x-www-form-urlencoded
```

Ukázka těla

```
Array
(
    [code] => 87f90ba9fe97e3b43f11c37eea1e1b475dbd59b9
    [redirect_uri] => https://mujeshopik.cz/stiteknabalik-endpoint/
    [scope] => deliveries collection-places
    [grant_type] => authorization_code
)
```

Description of request parameters (REQUEST)

Parameter	Example value	Description
`Authorization`	Basic empoeWdrbmtmazphYmNkMTIzNA==	HTTP Basic authentication of your application. Connect "Basic" and identifier of OAuth client with password, encoded as base64. Example: `"Basic ".base64_encode($client_id.":".$client_secret)`
`Content-type`	application/x-www-form-urlencoded	Always use this value. POST request body must containt values encoded as application/x-www-form-urlencoded.
`code`	87f90ba9fe97e3b43f11c37eea1e1b475dbd59b9	Authorization code received in previous response
`redirect_uri`	https://mujeshopik.cz/stiteknabalik-endpoint/	Registered redirect uri
`scope`	collection-places deliveries	List of requestd accesss scopes divided by space(each resource has its scope, more in method documentation)
`grant_type`	authorization_code	Always just **authorization_code**

Example of HTTP response for access token request (REQUEST)

Returned `access_token` is valid for 60 minutes (validity in seconds is in response) and `refresh_token` with unlimited expiration

Both tokens are 40 characters long and only contain lowercase characters and numbers. To obtain new access token you will receive same data structre without `refresh_token` item. In case that user denies your application access to his data all your tokens will be revoked

Detailed request description can be found in specification.

Ukázka hlavičky

```
HTTP/1.1 200
X-Frame-Options: SAMEORIGIN
Content-Type: application/json
Content-Encoding: gzip
```

Ukázka těla

```
{
    "access_token": "f709547842cb9f5b891bb9dd3dcaf90d512f8ddd",
    "expires_in": "3600",
    "token_type": "bearer",
    "scope": "deliveries collection-places",
    "refresh_token": "406aabf0e1aedc3c600cbf7f591e9b439408f5c3"
}
```

Step 4. - Access the resources

All resources require **HTTP Bearer authentication specified** in RFC 6750.
Use `access_token` in HTTP header Authorization along with **Bearer** prefix.

Using token in query or body of request is not supported.

Example request (REQUEST)

Endpoint otevřít

```
GET: https://rest.stiteknabalik.cz/v4/deliveries?deliveryId=15023456
```

Ukázka hlavičky

```
Authorization: Bearer f709547842cb9f5b891bb9dd3dcaf90d512f8ddd
Accept: application/json
```

Example of response with expired access token (RESPONSE)

It is necessary to check if you use valid non expired access token. If you request with expired or otherwise invalid token an error according to specification is returned. For expired token following response is returned:

Ukázka hlavičky

```
HTTP/1.1 401 Unauthorized
Authorization: Bearer f709547842cb9f5b891bb9dd3dcaf90d512f8ddd
WWW-Authenticate: Bearer realm="ClientApi", error="invalid_token", error_description="The access token provided has
expired"
Accept: application/json
```

Ukázka těla

```
{
    "code": "401",
    "status": "error",
    "message": "The access token provided has expired",
    "errors": []
}
```

Step 5. - access to resources - refresh access token

This step is used only when `access_tokenu` has expired.

Using `refresh_tokenu` your OAuth client requests new `access_tokenu` (valid for another 60 minutes)

Header of this POST request must contain identical params as when requesting using `authorization_code` (see. Step 3.). Body of request must contain `refresh_token`, `redirect_uri`, `scope`, `grant_type`.

Conten-type type should be **application/x-www-form-urlencoded**.

Description of parameters (REQUEST)

Parameter	Example value	Description
`Authorization`	Basic empoeWdrbmtmazphYmNkMTIzNA==	HTTP Basic authentication of your application. Connect "Basic" and identifier of OAuth client with password, encoded as base64. Example: `"Basic ".base64_encode($client_id.":".$client_secret)`
`Content-type`	application/x-www-form-urlencoded	Always use this value. POST request body must containt values encoded as application/x-www-form-urlencoded.
`refresh_token`	406aabf0e1aedc3c600cbf7f591e9b439408f5c3	Refresh token, kthat you received in `access_token` response
`redirect_uri`	https://mujeshopik.cz/stiteknabalik-endpoint/	Registered redirect uri
`scope`	collection-places deliveries	List of requestd accesss scopes divided by space(each resource has its scope, more in method documentation)
`grant_type`	refresh_token	Always just `refresh_token`

Example of refresh tokene request (REQUEST)

Endpoint

```
POST: https://rest.stiteknabalik.cz/oauth/token
```

Ukázka hlavičky

```
Authorization: empoeWdrbmtmazphYmNkMTIzNA==
Content-Type: application/x-www-form-urlencoded
```

Ukázka těla

```
Array
(
    [refresh_token] => 406aabf0e1aedc3c600cbf7f591e9b439408f5c3
    [redirect_uri] => https://mujeshopik.cz/stiteknabalik-endpoint/
    [scope] => deliveries collection-places
    [grant_type] => refresh_token
)
```

Ukázka odpovědi

Ukázka hlavičky

```
HTTP/1.1 201
Content-Type: application/json
Content-Encoding: gzip
```

Ukázka těla

```
{
    "access_token": "943a8490c94e6750789e2bbb0a0272b8d3833869",
    "expires_in": "3600",
    "token_type": "bearer",
    "scope": "deliveries collection-places"
}
```

© 2018 - 2025 Foxdeli s.r.o.,