Finoa API

The API server can be reached at https://demoapi.finoa.io. Only HTTPS connections are accepted.

The API offers a programmatical replacement for some of the functionality available on the Finoa website. It is intended for developers who wish to control assets stored with Finoa through their own user interface.

Versions

Different versions of the API can be selected by appending the desired version name to the server URL. The general scheme is `https://api.finoa.io/{version}`. Currently only one version is supported, namely `v1`.

Authentication

All requests to the API server require authentication. Valid authentication consists of the following four elements, each of which provides an HTTP request header:

User account authentication

To determine which Finoa account should be used, every request to the API must authenticate the user via HTTP basic access authentication. For this, the HTTP request must contain a header named `Authorization` which contains the keyword `Basic`, as well as username and password of the account concatenated with a semicolon `:` and then encoded via base64. For example, to access the user account `JohnDoe` with password `swordfish`, construct the string `JohnDoe:swordfish` and encode it in base64 to obtain `Sm9obkRvZTpzd29yZGZpc2g=`. The final header would look like this:

```
Authorization: Basic Sm9obkRvZTpzd29yZGZpc2g=
```

Timestamp

Every request needs to contain a valid HTTP `Date` header (see specification). Only requests with a timestamp not older than 60 seconds will be accepted by the server. An example would be:

```
Date: Wed, 06 Nov 2019 16:34:38 GMT
```

API account authentication

Not everyone with a valid Finoa account can use the API. To authenticate yourself as a valid API user, you need to provide your Finoa API key in a header named `Finoa-Api-Key`. For example, if your API key is `88cf528a-8ab9-44c1-8bd2-9af8d8542e79`, the final header would look like this:

```
Finoa-Api-Key: 88cf528a-8ab9-44c1-8bd2-9af8d8542e79
```

HMAC

To prevent modifications of the request, a HMAC-SHA256 digest of the request must be provided in the header `Finoa-Api-Digest`. The HMAC operation takes two inputs, a *secret* and a *message*. Together with your API key you should have received the base64 encoded *secret*. The *message* is constructed by concatenating the **timestamp** of the request, the **HTTP verb** used for the request, the **method** address of the request (including path and query parameters) and the request **body** if available.
Assuming your base64 encoded secret is `bXlTZWNyZXQ=` (which decodes to `mySecret`), the secret used in the HMAC operation would be `mySecret`. Note that for the sake of this example, the decoded secret is still a printable ASCII string. In general, the secret will consist of random bytes, which is why they are base64 encoded.
Furthermore, assume your timestamp is `Wed, 06 Nov 2019 16:34:38 GMT` and you want to call `PUT` on `/v1/example` with the following body:
`{"Currency": "BTC", "Info": "Example call"}`
then the message input to the HMAC operation would be
`Wed, 06 Nov 2019 16:34:38 GMTPUT/v1/example{"Currency": "BTC", "Info": "Example call"}`

The output of a SHA-256 HMAC on these values is `7a0333c05f5d7feea92e6307bd59625f092bfbd17d6180eb489812d10e9712f6`, so that the final header would look like this:

```
Fina-Api-Digest:
7a0333c05f5d7feea92e6307bd59625f092bfbd17d6180eb489812d10e9712f6
```

Be careful with whitespace (tab vs spaces and encoding of new line characters). The body used in the HMAC needs to be identical to the one sent with the API request.

Parameters

There are three ways that an API call can take parameters.

Path parameters

These are used for hierarchical arguments to the API method and are part of the method path. An example for a path parameter is the API version `v1` in a call to `https://api.finoa.io/v1/currencies`.

Query parameters

For safe requests (i.e. requests that don't change the server state) like GET, these are used for non-hierarchical arguments to the API method and are appended to the path with a question mark `?`. The parameter name and value are separated by an equal sign `=`. Multiple query parameters are separated by an ampersand `&`. Some query parameters can be used multiple times. For example, to query all Bitcoin and Ethereum addresses, you can call `https://api.finoa.io/v1/addresses?Currency=ETH&Currency=BTC`

Parameters in the JSON Body

For non-safe requests (i.e. requests that change the server state) like PUT and POST, parameters are usually given as name/value pairs in a JSON object stored in the request body.

Return values

Every request will return an HTTP status code and potentially a JSON object. The HTTP status code is one of the following:

Code	Explanation
200	The request was processed successfully
400	The request is invalid or malformed
401	The user account is not authorized
402	Insufficient funds to perform a transaction
403	Your API authentication is invalid
404	The method doesn't exist
405	The HTTP Verb is not supported on the endpoint
429	You have sent too many requests
500	An internal server error has occurred

If code 200 is returned, the result will also contain a JSON object consisting of a single name/value pair with the name `Result` and a value containing the result array as specified below.

Methods

This is a list of all methods supported by the API. They are listed by the relevant HTTP verb and the method path. Requests using GET are safe (i.e. they don't change the server state). PUT and POST requests change the server state and are usually called "non safe". The difference between PUT and POST is that PUT requests are "idempotent", which means that repeated application of the same PUT request will not change the server state further.

An example of an idempotent request is the addition of an address to the list of allowed destination addresses. Adding it once changes the server state (the list of allowed addresses), but adding it again will have no impact, because the address is already in the list.
POST requests are not idempotent, so they should only be submitted once. An example would be the submission of a transaction request. Sending the same request twice will result in two identical transactions being scheduled on the server.

`GET /{version}/addresses/{address}`

> **Description**
> Returns a list with details of all matching addresses for the given user account.
>
> **Path parameter**

- `version`
  Specifies the API version
- `address` [optional]
  Specifies a specific address

Query parameters

- `Currency` [optional - can be used more than once]
  Only show addresses for the specified currency
- `Limit` [optional]
  Limits the number of returned results to the specified amount
- `Offset` [optional]
  Skips the specified number of results.

Result
Each element in the list contains the following fields:

- `Currency` (*string*)
  Symbol for the currency of the address
- `Address` (*string*)
  The address
- `Balance` (*string*)
  The balance of this address in the given currency. To avoid truncation / rounding problems, this is given as a decimal string with a "." character as the decimal point.
- `Info` (*string*)
  An info string that was provided during the creation of this address

Example
`GET /v1/addresses?Currency=ETH` returns


```
[
    {
        "Currency": "ETH",
        "Address": "0xb794f5ea0ba39494ce839613fffba74279579268",
        "Balance": "12.123059",
        "Info": "My favourite address"
    },
    {
        "Currency": "ETH",
        "Address": "0x0472ec0185ebb8202f3d4ddb0226998889663cf2",
        "Balance": "0.02",
        "Info": "Some other address"
    }
]
```