Yandex.Disk API

Developer's guide

21.02.2018

Yandex

Yandex.Disk API. Developer's guide. Version 1.1
Document build date: 21.02.2018.
This volume is a part of Yandex technical documentation.
Yandex helpdesk site: http://help.yandex.ru

Copyright Disclaimer

Contact information

Yandex LLC
http://www.yandex.com
Phone: +7 495 739 7000
Email: pr@yandex-team.ru
Headquarters: 16 L'va Tolstogo St., Moscow, Russia 119021

Contents

Introduction

The Yandex.Disk API is a service providing an interface for accessing Yandex.Disk programmatically. The interface implements the requirements of the WebDAV protocol. The API can be used for downloading and uploading files, as well as managing files and directories.

The Yandex.Disk API is intended for applications that work with the files of Yandex.Disk users or store their own files and settings on Yandex.Disk.

The API documentation describes how to use the protocol methods to access Yandex.Disk and includes examples of API calls.

The documentation is intended for developers of applications that use Yandex.Disk features.

Getting started

The Yandex.Disk API is available at the address https://webdav.yandex.com. The connection must be made over the HTTPS protocol (port 443).

To access any user's data, the application must be authorized using Basic authentication or an OAuth access token.

Basic authentication

Applications can get access to users' Disks with usernames and passwords, following the Basic authentication mechanism of the HTTP protocol.

For Basic authentication, each of the application's requests to Yandex.Disk must contain the `Authorization` header in the following format:

```
Authorization: Basic <access token>
```

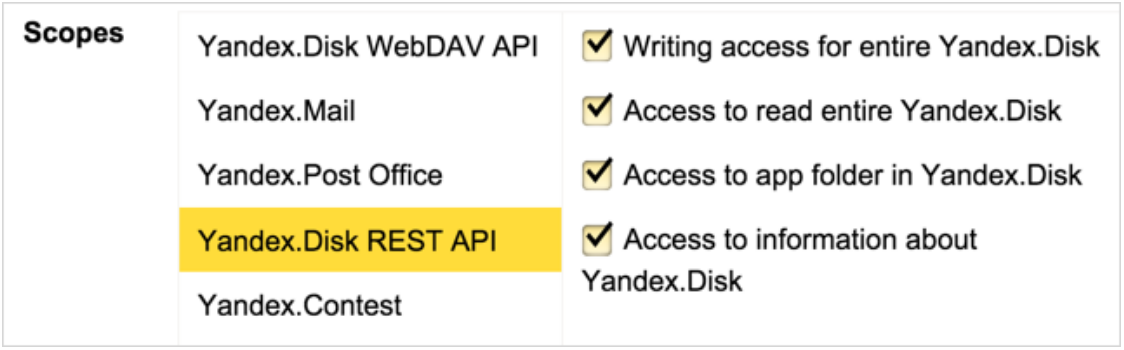
Here, the access token is a string in the format `<username>:<password>` in Base64 encoding.

Authorizing applications using OAuth tokens

Applications can get access to users' Disks by using OAuth tokens. Each access token allows a specific application to access the data of a specific user.

To access Yandex services using the OAuth protocol, the developer must register the application on the oauth.yandex.com/ service, in the Register new application section.

When registering an application that uses the Yandex.Disk API, you should select the appropriate access rights:



After registration, the application can use any appropriate method to get OAuth tokens for accessing user data.

The access token that is obtained must be passed in the `Authorization` header for every call to the Yandex.Disk API, indicating the token type before the token value. Example for this header:

```
"Authorization: OAuth 0c4181a7c2cf4521964a72ff57a34a07"
```

For information on obtaining and using OAuth access tokens, see the documentation for the Yandex OAuth service.

API methods

The Yandex.Disk features that are accessible using the API include uploading and downloading files, viewing the file structure, and managing the files and directories on Yandex.Disk.

All of these features are implemented using the methods of the WebDAV protocol:

- Uploading a file (PUT).
- Downloading a file (GET).
- Creating a directory (MKCOL).
- Copying (COPY).
- Moving and renaming (MOVE).
- Deleting (DELETE).
- Getting properties of files and directories (PROPFIND), including:
  - The amount of free and used space on Yandex.Disk.
  - The contents of directories.
- Changing the properties of a file or directory (PROPPATCH).

In addition to the above, the service nominally supports the LOCK and UNLOCK methods; the server returns a response that a file is either locked or unlocked, but in actuality, this does not affect the state of the file.

Uploading a file (PUT)

Use the PUT method to upload a file to Yandex.Disk.

At the beginning and end of uploading the file, the service checks whether the file exceeds the space available to the user on Disk. If there is not enough space, the service returns a response with the code 507 Insufficient Storage.

Support is provided for transferring compressed files (`Content-Encoding: gzip` header) and chunked files (`Transfer-Encoding: chunked`).

Checking for duplicate files

Users often want to upload files to Yandex.Disk that have already been uploaded by someone else. In such cases, Disk can just copy the needed file on the server, without uploading it.

The file is identified by the file size, MD5 checksum, and SHA-256 hash. The following headers are used for passing them:

```
Etag: <md5 checksum>
Sha256: <SHA-256 hash>
Content-Length: <file size in bytes>
```

If a duplicate file is found, the server responds with the code `201 Created`.

Example of uploading

The app uploads the `otpusk.avi` file to the `/a/` directory on the user's Disk, specifying the checksum and hash for checking for duplicates.

```
PUT /a/otpusk.avi HTTP/1.1
Host: webdav.yandex.ru
Accept: */*
Authorization: OAuth 0c4181a7c2cf4521964a72ff57a34a07
Etag: 1bc29b36f623ba82aaf6724fd3b16718
Sha256: T8A8H6B407D7809569CA9ABCB0082E4F8D5651E46D3CDB762D02D0BF37C9E592
Expect: 100-continue
Content-Type: application/binary
Content-Length: 103134024
```

If Yandex.Disk found a duplicate file and doesn't need to upload anything, the server responds with the code `201 Created`.

If a duplicate was not found, the server responds with the code `100 Continue`, allowing the file to be uploaded in the body of the next request. When the file has been uploaded successfully, the server also responds with the code `201 Created`.

Downloading a file (GET)

Use the GET method to download a file from Yandex.Disk.

The `Range` header can be used for requesting a particular section of the file. The response to this type of request contains the header `Content-Type: multipart/byteranges`.

To request a compressed file, add either the `TE: gzip,chunked` header to the request, or a combination of two headers:

```
TE: chunked
Accept-Encoding: gzip
```

The server will apply compression, if it is justifiable. The client must be prepared to handle both compressed and uncompressed responses.

The application downloads the `readme.pdf` file from the root directory of the user's Yandex.Disk.

```
GET /readme.pdf HTTP/1.1
Host: webdav.yandex.ru
Accept: */*
Authorization: OAuth 0c4181a7c2cf4521964a72ff57a34a07
```

If the file exists, the following response is returned:

```
HTTP/1.1 200 OK
Last-Modified: Mon, 09 Apr 2012 10:56:13 GMT
Etag: 2bf4a775cdaffe827bbad4998b9b09eb
Content-Length: 455833

<response body with the file>
```

Creating a directory (MKCOL)

Use the MKCOL method to create a new directory on Yandex.Disk.

According to the protocol, only one directory can be created as the result of a single request. If the application sends a request to create the `/a/b/c/` directory, but the `/a/` directory does not contain a `/b/` directory, the service will not create the `/b/` directory, and will respond with the code 409 Conflict.

The application creates the `/b/` directory inside the `/a/` directory, which is located in the Yandex.Disk root directory.

```
MKCOL /a/b/ HTTP/1.1
Host: webdav.yandex.ru
Accept: */*
Authorization: OAuth 0c4181a7c2cf4521964a72ff57a34a07
```

If the `/a/` directory exists and the `/b/` directory was created successfully, the following response is returned:

```
HTTP/1.1 201 Created
Content-Length: 0
```

Copying (COPY)

Use the COPY method to copy files and directories within the Yandex.Disk file structure.

Example: the`lion.png` file is copied from the `pictures` folder to the `animals` folder.

```
COPY /pictures/lion.png HTTP/1.1
Host: webdav.yandex.ru
Accept: */*
Authorization: OAuth 0c4181a7c2cf4521964a72ff57a34a07
Destination: /animals/lion.png
```

The `Overwrite` header can be set in order to prevent overwriting an existing file with the same name. The value "T" (by default) allows overwriting, and the value "F" forbids it. If the `/pictures/` directory already has the `lion.png` file, the request from the example will not be completed.

If copying was completed successfully, the following response is returned:

```
HTTP/1.1 201 Created
Content-Length: 0
```

Moving and renaming (MOVE)

Use the MOVE method to move or rename files and directories within the Yandex.Disk file structure.

Example: the `lion.png` file in the `pictures` folder is renamed to `kitty.png`.

```
MOVE /pictures/lion.png HTTP/1.1
Host: webdav.yandex.ru
Accept: */*
Authorization: OAuth 0c4181a7c2cf4521964a72ff57a34a07
Destination: /pictures/kitty.png
Overwrite: F
```

The `Overwrite` header can be set in order to prevent overwriting an existing file with the same name. The value "T" (by default) allows overwriting, and the value "F" forbids it. If the `/pictures/` directory already has the `kitty.png` file, the request from the example will not be completed.

If the move or rename was completed successfully, the following response is returned:

```
HTTP/1.1 201 Created
Content-Length: 0
```

Deleting (DELETE)

Use the DELETE method to delete a file or directory from Yandex.Disk.

According to the protocol, directories are always deleted together with all nested files and directories.

Deleting the `/pictures/` directory from the user's root Disk directory.

```
DELETE /pictures/ HTTP/1.1
Host: webdav.yandex.ru
Accept: */*
Authorization: OAuth 0c4181a7c2cf4521964a72ff57a34a07
```

If the directory was deleted successfully, the following response is returned:

```
HTTP/1.1 200 OK
Content-Length: 0
```

Getting properties of files and directories (PROPFIND)

Use the PROPFIND method to request:

- The amount of free/used space on Yandex.Disk (according to RFC 4331).
- A list of files and subdirectories contained in the directory (with the header `Depth: 1`).
- Other properties of a file or directory.

The list of all the properties supported in the context of the WebDAV protocol is provided in the DAV Properties section of the protocol description. The XML elements that are used in requests and responses for properties are documented in the protocol specification.

Examples of PROPFIND method requests are provided in the sections:

- Free and used space.
- Directory contents.
- Properties of a file or directory.

Free and used space

To find out how much space is in use on Yandex.Disk and how much free space is left, send a PROPFIND request in any directory, specifying the corresponding properties in the request body:

- `quota-available-bytes` — Free space.
- `quota-used-bytes` — Used space.

Request for the amount of free and used space:

```
PROPFIND / HTTP/1.1
Host: webdav.yandex.ru
Accept: */*
Depth: 0
Authorization: OAuth 0c4181a7c2cf4521964a72ff57a34a07

<D:propfind xmlns:D="DAV:">
  <D:prop>
    <D:quota-available-bytes/>
    <D:quota-used-bytes/>
  </D:prop>
</D:propfind>
```

The service returns the requested properties, specifying the amount of free and used space on Yandex.Disk in bytes:

```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: 320

<?xml version="1.0" encoding="utf-8"?>
<d:multistatus xmlns:d="DAV:">
   <d:response>
      <d:href>/</d:href>
      <d:propstat>
         <d:status>HTTP/1.1 200 OK</d:status>
         <d:prop>
            <d:quota-available-bytes>282476624607</d:quota-available-bytes>
            <d:quota-used-bytes>4212442401</d:quota-used-bytes>
         </d:prop>
      </d:propstat>
   </d:response>
</d:multistatus>
```

Directory contents

To get a list of files and subdirectories, specify the `Depth` header in the request with the value “1”. In the response, Yandex.Disk returns the directory properties along with all the items that are located at the top level of the directory.

To get a paginated list of nested elements, set the number of items to skip (the `offset` parameter) and the desired number of items per page (the `amount` parameter). It is assumed that the items are arranged alphabetically, and any nested directories are listed before the files.

Request for complete directory contents

The application requests properties of the Yandex.Disk root directory and the items directly contained in it:

```
PROPFIND / HTTP/1.1
Host: webdav.yandex.ru
Accept: */*
Depth: 1
Authorization: OAuth 0c4181a7c2cf4521964a72ff57a34a07
```

The response describes all the items contained in the directory and their properties:

```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: 3079

<?xml version="1.0" encoding="utf-8"?>
<d:multistatus xmlns:d="DAV:">
  <d:response>
    <d:href>
      /
    </d:href>
    <d:propstat>
      <d:status>
        HTTP/1.1 200 OK
      </d:status>
      <d:prop>
        <d:creationdate>
          1970-01-01T00:00:00Z
        </d:creationdate>
        <d:displayname>
          disk
        </d:displayname>
        <d:getcontentlength>
          0
        </d:getcontentlength>
        <d:getlastmodified>
          Thu, 01 Jan 1970 00:00:00 GMT
        </d:getlastmodified>
        <d:resourcetype>
          <d:collection/>
        </d:resourcetype>
      </d:prop>
    </d:propstat>
  </d:response>
  <d:response>
    <d:href>
      /Documents/
    </d:href>
    <d:propstat>
      <d:status>
        HTTP/1.1 200 OK
      </d:status>
      <d:prop>
        <d:creationdate>
          2012-03-24T09:00:43Z
        </d:creationdate>
        <d:displayname>
          Documents
        </d:displayname>
        <d:getcontentlength>
          0
        </d:getcontentlength>
        <d:getlastmodified>
```

```
          Sat, 24 Mar 2012 09:00:43 GMT
        </d:getlastmodified>
        <d:resourcetype>
          <d:collection/>
        </d:resourcetype>
      </d:prop>
    </d:propstat>
  </d:response>
  <d:response>
    <d:href>
      /readme.pdf
    </d:href>
    <d:propstat>
      <d:status>
        HTTP/1.1 200 OK
      </d:status>
      <d:prop>
        <d:creationdate>
          2012-04-09T10:56:13Z
        </d:creationdate>
        <d:displayname>
          readme.pdf
        </d:displayname>
        <d:getcontentlength>
          455833
        </d:getcontentlength>
        <d:getcontenttype>
          application/pdf
        </d:getcontenttype>
        <d:getlastmodified>
          Mon, 09 Apr 2012 10:56:13 GMT
        </d:getlastmodified>
        <d:resourcetype/>
      </d:prop>
    </d:propstat>
  </d:response>
</d:multistatus>
```

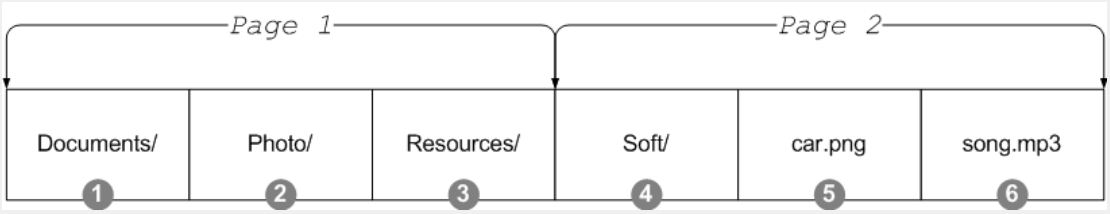
Request for paginated directory contents

Each of the anticipated pages should contain three items.

To request the first page, just pass the `amount` parameter with the value “3”. To request the second page, the first three items should be skipped; to do this, set the `offset` parameter with the value “3”.

```
PROPFIND /Downloads/?offset=3&amount=3 HTTP/1.1
Host: webdav.yandex.ru
Accept: */*
Depth: 1
Authorization: OAuth 0c4181a7c2cf4521964a72ff57a34a07
```

The response shows the requested directory and three items, numbered 4, 5, and 6:




```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: 1737

<?xml version="1.0" encoding="utf-8"?>
<d:multistatus xmlns:d="DAV:">
  <d:response>
    <d:href>/Downloads/</d:href>
    <d:propstat>
      <d:status>HTTP/1.1 200 OK</d:status>
      <d:prop>
        <d:resourcetype>
          <d:collection/>
        </d:resourcetype>
```

```
        <d:getlastmodified>Wed, 04 Apr 2012 20:00:00 GMT</d:getlastmodified>
        <d:getcontentlength>0</d:getcontentlength>
        <d:displayname>Downloads</d:displayname>
        <d:creationdate>2012-04-04T20:00:00Z</d:creationdate>
      </d:prop>
    </d:propstat>
  </d:response>

  <d:response>
    <d:href>/Downloads/Soft/</d:href>
    <d:propstat>
      <d:status>HTTP/1.1 200 OK</d:status>
      <d:prop>
        <d:resourcetype>
          <d:collection/>
        </d:resourcetype>
        <d:getlastmodified>Wed, 25 Jul 2012 12:23:22 GMT</d:getlastmodified>
        <d:getcontentlength>0</d:getcontentlength>
        <d:displayname>Soft</d:displayname>
        <d:creationdate>2012-07-25T12:23:21Z</d:creationdate>
      </d:prop>
    </d:propstat>
  </d:response>

  <d:response>
    <d:href>/Downloads/car.png</d:href>
    <d:propstat>
      <d:status>HTTP/1.1 200 OK</d:status>
      <d:prop>
        <d:resourcetype/>
        <d:getlastmodified>Wed, 25 Jul 2012 12:23:57 GMT</d:getlastmodified>
        <d:getcontenttype>image/png</d:getcontenttype>
        <d:getcontentlength>63434</d:getcontentlength>
        <d:displayname>car.png</d:displayname>
        <d:creationdate>2012-07-25T12:23:56Z</d:creationdate>
      </d:prop>
    </d:propstat>
  </d:response>

  <d:response>
    <d:href>/Downloads/song.mp3</d:href>
    <d:propstat>
      <d:status>HTTP/1.1 200 OK</d:status>
      <d:prop>
        <d:resourcetype/>
        <d:getlastmodified>Wed, 25 Jul 2012 12:23:57 GMT</d:getlastmodified>
        <d:getcontenttype>audio/mpeg</d:getcontenttype>
        <d:getcontentlength>6343431</d:getcontentlength>
        <d:displayname>song.mp3</d:displayname>
        <d:creationdate>2012-07-25T12:23:56Z</d:creationdate>
      </d:prop>
    </d:propstat>
  </d:response>
</d:multistatus>
```

Properties of a file or directory

The section Changing the properties of a file or directory (PROPPATCH) describes how properties are created.

If the request body does not indicate specific properties, Yandex.Disk sends the following in the response:

- Date and time the object was modified.
- Whether the object is a directory.
- For files, the size and type of contents.

Request for the value of the `myprop` property for the `/a/` directory:

```
PROPFIND /a/ HTTP/1.1
Host: webdav.yandex.ru
Accept: */*
Depth: 1
Authorization: OAuth 0c4181a7c2cf4521964a72ff57a34a07
Content-Length: 115
Content-Type: application/x-www-form-urlencoded

<?xml version="1.0" encoding="utf-8" ?>
<propfind xmlns="DAV:">
  <prop>
    <myprop xmlns="mynamespace"/>
  </prop>
</propfind>
```

If the directory exists and it has this property, the following response is returned:

```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: 252

<?xml version="1.0" encoding="utf-8"?>
<d:multistatus xmlns:d="DAV:">
  <d:response>
    <d:href>/a/</d:href>
    <d:propstat>
      <d:status>HTTP/1.1 200 OK</d:status>
      <d:prop>
        <myprop xmlns="mynamespace">
          myvalue
        </myprop>
      </d:prop>
    </d:propstat>
  </d:response>
</d:multistatus>
```

Changing the properties of a file or directory (PROPPATCH)

Use the PROPPATCH method to change the properties of a file or directory on Yandex.Disk. Applications can create their own custom properties for files and directories, for storing metainformation. Values of created properties can be requested using the PROPFIND method (see Properties of a file or directory).

The XML elements that are used in requests and responses are documented in the protocol specification.

For the `/a/` directory, the `myprop` property is created with the `myvalue` value.

```
PROPPATCH /a/ HTTP/1.1
Host: webdav.yandex.ru
Accept: */*
Authorization: OAuth 0c4181a7c2cf4521964a72ff57a34a07
Content-Length: 159
Content-Type: application/x-www-form-urlencoded

<?xml version="1.0" encoding="utf-8" ?>
<propertyupdate xmlns="DAV:" xmlns:u="mynamespace">
  <set><prop>
    <u:myprop>myvalue</u:myprop>
  </prop></set>
</propertyupdate>
```

Request response:

```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: 235

<?xml version="1.0" encoding="utf-8"?>
<d:multistatus xmlns:d="DAV:">
  <d:response>
    <d:href>/</d:href>
    <d:propstat>
      <d:status>HTTP/1.1 200 OK</d:status>
      <d:prop>
        <myprop xmlns="mynamespace"/>
      </d:prop>
    </d:propstat>
  </d:response>
</d:multistatus>
```

Publishing files and folders

Files and folders that are uploaded to Yandex.Disk can be published by generating a link that makes them accessible to people other than the Yandex.Disk owner.

Published files and folders can be reverted to private status, and the generated public links to the file will stop working.

Use the `PROPPATCH` method to set and change the "public" status. To publish a file or folder, assign any non-empty value to the `public_url` property in the `urn:yandex:disk:meta` namespace.

Example of publishing a folder

Publishing the `/public_folder/` folder located in the root directory of the user's Disk.

```
PROPPATCH /public_folder/ HTTP/1.1
User-Agent: my_application/0.0.1
Host: webdav.yandex.ru
Authorization: OAuth 0c4182a7c2cf4521964a72ff57a34a07
Content-Length: 158

<propertyupdate xmlns="DAV:">
  <set>
    <prop>
      <public_url xmlns="urn:yandex:disk:meta">true</public_url>
    </prop>
  </set>
</propertyupdate>
```

If the folder was published successfully, the server responds with:

```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: 390

<?xml version="1.0" encoding="utf-8"?>
<d:multistatus xmlns:d="DAV:">
  <d:response>
    <d:href>/public_folder/</d:href>
    <d:propstat>
      <d:status>HTTP/1.1 200 OK</d:status>
      <d:prop>
        <public_url xmlns="urn:yandex:disk:meta"> http://yadi.sk/d/FTb3fLiI49Xt0 </
public_url>
      </d:prop>
    </d:propstat>
  </d:response>
</d:multistatus>
```

The link to the published folder is returned in the value of the `public_url` element. If you attempt to publish a folder that is already published, Yandex.Disk returns the link that was generated during the first publication.

Example of publishing a file

Publishing the `readme.txt` file located in the `/public_folder/` directory on the user's Disk.

```
PROPPATCH /public_folder/readme.txt HTTP/1.1
User-Agent: my_application/0.0.1
Host: webdav.yandex.ru
Authorization: OAuth 0c4182a7c2cf4521964a72ff57a34a07
Content-Length: 158

<propertyupdate xmlns="DAV:">
  <set>
    <prop>
      <public_url xmlns="urn:yandex:disk:meta">true</public_url>
    </prop>
  </set>
</propertyupdate>
```

If the file was published successfully, the server responds with:

```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: 400

<?xml version="1.0" encoding="utf-8"?>
<d:multistatus xmlns:d="DAV:">
  <d:response>
    <d:href>/public_folder/readme.txt</d:href>
    <d:propstat>
      <d:status>HTTP/1.1 200 OK</d:status>
      <d:prop>
        <public_url xmlns="urn:yandex:disk:meta"> http://yadi.sk/d/UDh3fLiI49Xt0 </
public_url>
      </d:prop>
    </d:propstat>
  </d:response>
</d:multistatus>
```

The link to the published file is returned in the value of the `public_url` element. If you attempt to publish a file that is already published, Yandex.Disk returns the link that was generated during the first publication.

Example of unpublishing a published file

The `readme.txt` file in the `/public_folder/` catalog on the user's Disk is switched to “private” status — the `public_url` property is deleted.

```
PROPPATCH /public_folder/readme.txt HTTP/1.1
User-Agent: my_application/0.0.1
Host: webdav.yandex.ru
Authorization: OAuth 0c4182a7c2cf4521964a72ff57a34a07
Content-Length: 149

<propertyupdate xmlns="DAV:">
  <remove>
    <prop>
      <public_url xmlns="urn:yandex:disk:meta" />
    </prop>
  </remove>
</propertyupdate>
```

In the response, Yandex.Disk confirms that the property value is now empty:

```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: 336

<?xml version="1.0" encoding="utf-8"?>
<d:multistatus xmlns:d="DAV:">
  <d:response>
    <d:href>/public_folder/readme.txt</d:href>
    <d:propstat>
      <d:status>HTTP/1.1 200 OK</d:status>
      <d:prop>
        <public_url xmlns="urn:yandex:disk:meta" />
      </d:prop>
    </d:propstat>
  </d:response>
</d:multistatus>
```

If you attempt to unpublish a file that was not published, the server response is the same — Yandex.Disk informs you that the property value is empty.

Example of checking public status

Checking whether the `/public_folder/` directory is public: using the `PROPFIND` method, we request the value of the `public_url` property.

```
PROPFIND /public_folder/ HTTP/1.1
User-Agent: my_application/0.0.1
Host: webdav.yandex.ru
Authorization: OAuth 0c4182a7c2cf4521964a72ff57a34a07
Depth: 0
Content-Length: 105

<propfind xmlns="DAV:">
  <prop>
    <public_url xmlns="urn:yandex:disk:meta"/>
  </prop>
</propfind>
```

If the folder is published, the server returns a public link in the property value:

```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: 390

<?xml version="1.0" encoding="utf-8"?>
<d:multistatus xmlns:d="DAV:">
  <d:response>
    <d:href>/public_folder/</d:href>
    <d:propstat>
      <d:status>HTTP/1.1 200 OK</d:status>
      <d:prop>
        <public_url xmlns="urn:yandex:disk:meta">
           http://yadi.sk/d/FTb3fLiI49Xt0
        </public_url>
      </d:prop>
    </d:propstat>
  </d:response>
</d:multistatus>
```

If the folder is not published, the server reports that the property value is empty:

```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: 340

<?xml version="1.0" encoding="utf-8"?>
<d:multistatus xmlns:d="DAV:">
  <d:response>
    <d:href>/public_folder/</d:href>
    <d:propstat>
      <d:status>HTTP/1.1 404 Object Not Found</d:status>
      <d:prop>
        <public_url xmlns="urn:yandex:disk:meta" />
      </d:prop>
    </d:propstat>
  </d:response>
</d:multistatus>
```

Getting image previews

You can request a preview (a smaller version of a picture) for images that have been uploaded to Yandex.Disk.

To get an image preview, send a GET request specifying the `preview` parameter, and set the desired size as the value of the `size` parameter. There are several ways to set the preview size:

- T-shirt size (like in Yandex.Fotki), such as `size=M`.

  Yandex.Disk returns a preview in the size you selected.

  Supported values:

  - “XXXS” — 50 pixels on each side (square).
  - “XXS” — 75 pixels on each side (square).
  - “XS” — 100 pixels on each side (square).
  - “S” — 150 pixels wide, preserves original aspect ratio.
  - “M” — 300 pixels wide, preserves original aspect ratio.
  - “L” — 500 pixels wide, preserves original aspect ratio.
  - “XL” — 800 pixels wide, preserves original aspect ratio.
  - “XXL” — 1024 pixels wide, preserves original aspect ratio.
  - “XXXL” — 1280 pixels wide, preserves original aspect ratio.
- A number, such as `size=128`.

  Yandex.Disk returns a preview with this width. If the specified width is more than 100 pixels, the preview preserves the aspect ratio of the original image.

Otherwise, the preview is additionally modified: the largest possible square section is taken from the center of the image to scale to the specified width.

- Exact dimensions, such as `size=128x256`.

  Yandex.Disk returns a preview with the specified dimensions. The largest possible section with the specified width/height ratio is taken from the center of the original image (in the example, the ratio is 128/256 or 1/2). Then this section of the image is scaled to the specified dimensions. See the example with exact dimensions below.

- Exact width or height, such as `size=128x` or `size=x256`.

  Yandex.Disk returns a preview with the specified width or height that preserves the aspect ratio of the original image.

Server response

There are two possible server responses:

- HTTP 404 — The image was not found on Yandex.Disk.
- HTTP 200 — The image was found, and the preview was created and returned in the response body.

Example requests

The examples use an image sized 1280×720 pixels that has been uploaded to the `/fotki/` directory on the user's Disk.

The headings correspond to the value of the `size` parameter for each example.

XS

Requests an XS-size preview (square with 100 pixels per side).

```
GET /fotki/t.jpg?preview&size=XS HTTP/1.1
User-Agent: my_application/0.0.1
Host: webdav.yandex.ru
Authorization: OAuth 0c4182a7c2cf4521964a72ff57a34a07
```

The server responds with a preview file sized 100×100 pixels:

XL

Requests an XL-size preview (width of 800 pixels and the original aspect ratio).

```
GET /fotki/t.jpg?preview&size=XL HTTP/1.1
User-Agent: my_application/0.0.1
Host: webdav.yandex.ru
Authorization: OAuth 0c4182a7c2cf4521964a72ff57a34a07
```

The server responds with a preview file sized 800×533 pixels:

90

Requests a preview 90 pixels wide.

```
GET /fotki/t.jpg?preview&size=90 HTTP/1.1
User-Agent: my_application/0.0.1
Host: webdav.yandex.ru
Authorization: OAuth 0c4182a7c2cf4521964a72ff57a34a07
```

The server responds with a preview file sized 90×90 pixels:

100

Requests a preview 100 pixels wide.

```
GET /fotki/t.jpg?preview&size=100 HTTP/1.1
User-Agent: my_application/0.0.1
Host: webdav.yandex.ru
Authorization: OAuth 0c4182a7c2cf4521964a72ff57a34a07
```

The server responds with a preview file sized 100×100 pixels:

101

Requests a preview 101 pixels wide.

```
GET /fotki/t.jpg?preview&size=101 HTTP/1.1
User-Agent: my_application/0.0.1
Host: webdav.yandex.ru
Authorization: OAuth 0c4182a7c2cf4521964a72ff57a34a07
```

The server responds with a preview file sized 101×67 pixels:

90x

Requests a preview with a width of 90 pixels that preserves the original aspect ratio.

```
GET /fotki/t.jpg?preview&size=90x HTTP/1.1
User-Agent: my_application/0.0.1
Host: webdav.yandex.ru
Authorization: OAuth 0c4182a7c2cf4521964a72ff57a34a07
```

The server responds with a preview file sized 90×60 pixels:

700x1000

Requests a preview with the exact dimensions of 700×1000 pixels.

```
GET /fotki/t.jpg?preview&size=700x1000 HTTP/1.1
User-Agent: my_application/0.0.1
Host: webdav.yandex.ru
Authorization: OAuth 0c4182a7c2cf4521964a72ff57a34a07
```

The largest possible section is taken from the center of the picture with a width-to-height ratio of 7:10 (700 to 1000). In this case, the size of the fragment is 504×720.

Then this fragment is scaled to the specified dimensions. The server responds with a preview file sized 700×1000 pixels:

Request for user login

You can request the login of the user whose OAuth token you are using for authentication. This method is not available for Basic authentication.

The login of the user with the specified access token is requested.

```
GET /?userinfo HTTP/1.1
Host: webdav.yandex.com
Accept: */*
Authorization: OAuth 0c4182a7c2cf4521964a72ff57a34a07
```

If the passed token is valid, the following response is returned:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 10

login:test
```

The response body contains the `login` field, where the value “test” is the requested login.

Notification of changes on Yandex.Disk

Yandex.Disk can send the client application notification of changes on a user's Disk: files uploaded or deleted, directories created or deleted, and changes to the amount of space available.

Notifications are implemented using the XMPP protocol: the application connects to the Yandex.Disk server and gets messages in a specific format.

Connecting to the XMPP server

Establishing the server connection

The application must connect to port 5222 on the server `push.xmpp.yandex.ru`.

If port 5222 is not available, port 443 can be used instead; for this port, an encrypted connection is required, so the server connection must start with a TLS handshake (step 3).

Steps for connecting to the XMPP server:

1. Handshake.

   The application requests to create a stream:

   ```
   <?xml version="1.0"?>
   <stream:stream xmlns:stream="http://etherx.jabber.org/streams" version="1.0"
   xmlns="jabber:client" to="ya.ru" xml:lang="en" xmlns:xml="http://www.w3.org/
   XML/1998/namespace">
   ```

   The server returns the stream ID and information about available features (encryption, ZLIB compression, and SASL authorization):

   ```
   <?xml version='1.0'?>
   <stream:stream xmlns='jabber:client' xmlns:stream='http://etherx.jabber.org/
   streams' id='4235063168' from='ya.ru' version='1.0' xml:lang='en'>

   <stream:features>
     <starttls xmlns="urn:ietf:params:xml:ns:xmpp-tls"/>
     <compression xmlns="http://jabber.org/features/compress">
       <method>zlib</method>
     </compression>
     <mechanisms xmlns="urn:ietf:params:xml:ns:xmpp-sasl"/>
   </stream:features>
   ```

2. Establishing an encrypted connection (TLS).

   The application sends the encryption request stanza:

   ```
   <starttls xmlns="urn:ietf:params:xml:ns:xmpp-tls"/>
   ```

   The server confirms the encrypted connection:

   ```
   <proceed xmlns="urn:ietf:params:xml:ns:xmpp-tls"/>
   ```

3. TLS handshake. The application repeats the request to create a stream (see step 1), using an encrypted connection.
4. If necessary, the application may request to compress the stream in ZLIB format. In the XMPP protocol, compression is described as an extension of the standard protocol: XEP-0138.
5. Authorization. The application passes the server a Base64-encoded authorization token. The token must be formed from the user's login and the OAuth token, separating them with a zero byte, for example: `test\00c4181a7c2cf4521964a72ff57a34a07`.

```
<auth xmlns="urn:ietf:params:xml:ns:xmpp-sasl" mechanism="X-YANDEX-
OAUTH">dGVzdABjNDE4MWE3YzJjZjQ1MjE5NjRhNzJmZjU3YTM0YTA3</auth>
```

When authorization is successful, the server responds with:

```
<success xmlns="urn:ietf:params:xml:ns:xmpp-sasl"/>
```

6. Authorized handshake. The application requests to create a stream (see step 1). The server lists the features that are available after authorization.
7. Setting up the resource. The application passes the resource name in the following stanza:

```
<iq type="set" id="bind_1">
  <bind xmlns="urn:ietf:params:xml:ns:xmpp-bind">
    <resource>YaDisk-client</resource>
  </bind>
</iq>
```

The server returns the JID assigned to the application, which should be used in the following requests:

```
<iq xmlns="jabber:client" type="result" id="bind_1">
  <bind xmlns="urn:ietf:params:xml:ns:xmpp-bind">
    <jid>test@ya.ru/YaDisk-client</jid>
  </bind>
</iq>
```

8. Opening a session. The application requests to open a session:

```
<iq type="set" id="session_1">
  <session xmlns="urn:ietf:params:xml:ns:xmpp-session"/>
</iq>
```

The server responds with the result of opening the session:

```
<iq type="result" id="session_1">
<session xmlns="urn:ietf:params:xml:ns:xmpp-session"/>
</iq>
```

9. Request for the client version (XEP-0092).

After the session is opened, the server requests the version of the client application:

```
<iq from="ya.ru" type="get" to="test@ya.ru/YaDisk-client" id="ask_version">
<query xmlns="jabber:iq:version"/>
</iq>
```

The application should return its own name, the application version number and, optionally, the OS version:

```
<iq type="result" to="ya.ru" id="ask_version">
  <query xmlns="jabber:iq:version">
    <name>YaDiskClient</name>
    <version>1.1</version>
    <os>Mac OS X</os>
  </query>
</iq>
```

After the connection has been established, the server periodically checks the connection with the application by sending the following types of requests:

```
<iq from="ya.ru" type="get" to="test@ya.ru/YaDisk-client" id="ping_1">
  <ping xmlns="urn:xmpp:ping"/>
</iq>
```

The application should respond in the following way, copying the value of the `id` attribute from the server request to its response:

```
<iq type="result" to="ya.ru" id="ping_1"/>
```

Closing a connection

When finished, the application should close the XMPP stream by sending the server the closing string:

```
</stream:stream>
```

Getting notifications

To subscribe to notifications of changes on Yandex.Disk, the application sends the following type of request:

```
<iq to="test@ya.ru" type="set" id="13">
  <s xmlns="yandex:push:disk"/>
</iq>
```

When subscription is successful, the server responds with:

```
<iq from="test@ya.ru" type="result" to="test@ya.ru/YaDisk-client" id="13"/>
```

After subscribing successfully, the server sends notifications about changes on the user's Disk in this XMPP stream. Each notification contains the following type of stanza:

```
<iq from="test@ya.ru" type="set" to="test@ya.ru/YaDisk-client">
  <query xmlns="yandex:push:disk">
    <diff new="1343938312551221" old="134393831251560 5">
      <op type="new" key="/disk/file.doc" folder="/disk/"
md5="91bb2691d5d0cdc966e29798db7266c9"
sha256="f275425e3b38260a3693aad61c6f6aecabed2ca6a301426ad22fa7c64dae7a21"
size="14271" />
      <op type="deleted" key="/disk/oldfile.doc" folder="/disk/" />
      <op type="changed" key="/disk/changedfile.doc" folder="/disk/"
md5="f21e5e336227a21ab5d0e3fb790a1565"
sha256="c229b9fe30122ccee5790cb5135e41efa6910434bba14145ff83fb4fd19835f8"
size="13463" />
      <op type="new" key="/disk/folder/" folder="/disk/" />
      <op folder="/disk/Drivers/" type="published" key="/disk/Drivers/file1.zip"/
>
      <op folder="/disk/Drivers/" type="unpublished" key="/file2.zip"/>
    </diff>
    <quota available="8581196091" used="8738501"/>
  </query>
</iq>
```

Changes that occur on the user's Disk are described in the `<query>` element. Elements that can be included in `<query>`:

diff
Description of changes that occurred on the user's Disk.

After each change, Yandex.Disk registers a new version of the file system. The `<diff>` element describes the differences between two specific versions. The numbers of the versions being compared are specified in the element attributes.

Element attributes:
old
The number of the previous version of the Yandex.Disk file structure.
new
The number of the current version of the Yandex.Disk file structure.
Nested elements:
op
Description of a specific change on the Disk.

Changes registered:

- Creation, modification, and deletion of files.
- Creation and deletion of directories.

Element attributes:

type
Type of change. Possible values:

- “new” — File or directory was created.
- “deleted” — File or directory was deleted.
- “changed” — File was modified.
- “published” — File was published.
- “unpublished” — File was made private.

key
The path to the modified file or directory, relative to the root directory on Yandex.Disk.

folder
The directory on Yandex.Disk where the modified file or directory resides.

md5
Checksum for the file, calculated using the MD5 algorithm.

sha256
Checksum for the file, calculated using the SHA256 algorithm.

size
File size.

quota
Data about the used and available space on the user's Disk.
Element attributes:

available
Space available on the Disk, in bytes.

used
Space used on the Disk, in bytes.

Yandex

Yandex.Disk API

Developer's guide

21.02.2018