NFON Service Portal API Usage Manual

A. General Information

Table of Contents

Version History

DATE	CHANGES
29.11.2023	Updated and corrected Examples Added Signature Script
15.03.2024	Updated Script: correct API-URL and parameter encoding fix

1. Introduction

The NFON Service Portal Application Programming Interface (API) is currently available at the download area.

The NFON API is available as a RESTful web service. NFON provides multiple APIs. The Service Portal API 3 (SP3) is tailored to an Account-K level.

Some Use Cases to illustrate the power of the Service Portal API (SPA).

- Mass extensions deployment
- Active Directory interoperability
- Interactive creation and management of conference bridges
- Other third-party management tools to automatically administer the PBX Account-K

Before looking into the NFON REST Web Services (RESTful APIs), let's address why we do it that way. Over the past decades many methods were seen to get computers connected via a network, using the same language. The evolution of these methods and protocols resulted in multiple agreed standards, each addressing the communication in a specific way. At the top are the defined Application and Presentation layers. The REST Web Services are as well at the top layer and make it easy for humans to understand what's being exchanged while allowing computers to communicate efficiently. The advantage of REST Web Services is the minimum amount of data used on the same known mechanisms that define the WWW. Thinner than other protocols (e.g. SOAP).

The REST Web Services work almost exactly like a website in a browser, see our demo Client available as chrome extension: api-client.crx. The client can access a URL and receive data about the resource.

2. Used symbols

Used icons/characters.

Type	Icon/Character	Description
NOTE CAUTION WARNING	⚠	NOTE: Information that is useful but not critical to the reader. CAUTION: Tells the reader to proceed with caution. WARNING: Stronger than CAUTION; means "don't do this" or that this step could be irreversible, e.g., result in permanent data loss.
TIP	💡	Useful tips that provide additional information.

Procedure	1. 2. 3.	Procedures are numbered steps to perform an action, where the order of the steps is relevant.
Result of procedure/action	⇒	Indicates the result of an action//procedure.
Lists	· · ·	Used for listings and items where the order of steps is irrelevant.

3. Important Notes

The endpoints may have been changed! Please download the latest API Endpoints documentation here.

Download

B. Function

1. Introduction

The NFON API fully follows the REST paradigm "Hypermedia as the Engine of Applicate State" (HATEOAS), which basically means that all representations of resources shall be linked to each other in a similiar way how hypertext documents work in a browser.

https://en.wikipedia.org/wiki/HATEOAS

The NFON API is accessed through a REST web service interface over an https-connection. REST is an abbreviation for REpresentational State Transfer.

https://en.wikipedia.org/wiki/Representational_state_transfer

The HTTP verbs GET, POST, PUT and DELETE are used to query, create, update and delete resources.

https://www.rfc-editor.org/rfc/rfc9110.html

HTTP OPTIONS may be used to query which operations a resource or a collection list of resources will support.

https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

The HTTP verb HEAD may be used to check whether a specific resource exists. The HTTP body of the response will be empty.

https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

In Javascript Object Notation (JSON). JSON is considered the "fat-free" alternative to XML. It presents data in a compact format, easily readable by humans, and makes it possible to adopt the API in any Web browser. This still allows to present the resources XML encoded at a later time.

https://en.wikipedia.org/wiki/JSON

2. Signing and Authentication REST Requests

For signing and authentication REST Requests please contact the NFON support team to create a ticket in order to receive the necessary keys.

This topic explains authenticating requests using Apikey specification for NFON.

Authentication is the process of proving your identity to the system. Identity is an important factor in NFON access control decisions. Requests are allowed or denied in part based on the identity of the requester. For example, the right to create buckets is reserved for registered developers and (by default) the right to create objects in a bucket is reserved for the owner of the bucket in question. As a developer, you'll be making requests that invoke these privileges, so you'll need to prove your identity to the system by authenticating your requests. This section shows you how.

The NFON REST API uses a custom HTTP scheme based on a keyed-HMAC (Hash Message Authentication Code) for authentication.

To authenticate a request, you first concatenate selected elements of the request to form a string. You then use your NFON secret access key to calculate the HMAC of that string. Informally, we call this process "signing the request," and we call the output of the HMAC algorithm the signature, because it simulates the security properties of a real signature. Finally, you add this signature as a parameter of the request by using the syntax described in this section.

When the system receives an authenticated request, it fetches the NFON secret access key that you claim to have and uses it in the same way to compute a signature for the message it received. It then compares the signature it calculated against the signature presented by the requester. If the two signatures match, the system concludes that the requester must have access to the NFON secret access key and therefore acts with the authority of the principal to whom the key was issued. If the two signatures do not match, the request is dropped, and the system responds with an error message.

Example Authenticated NFON API REST Request

(note that this request is only to give an example but does not actually exist in reality.)

GET /photos/puppy.jpg HTTP/1.1 Host:

webservice.nfon.net

Date: Mon, 26 Mar 2007 19:37:58 +0000

Authorization: NFON-API AKIAIOSFODNN7EXAMPLE:frJIUN8DYpKDtOLCwo//yllqDzg=

3. The Authentication Header

The NFONREST API uses the standard HTTP Authorization header to pass authentication information. (The name of the standard header is unfortunate because it carries authentication information, not authorization.) Under the NFON authentication scheme, the Authorization header has the following form:

Authorization: NFON-API ApiAccessKeyId:Signature

Developers are issued an NFON access key ID and NFON secret access key when they register. For request authentication, the NFON **AccessKeyId** element identifies the access key ID that was used to compute the **signature** and, indirectly, the developer making the request.

The **signature** element is the RFC 2104 HMAC-SHA1 of selected elements from the request, and so the **signature** part of the Authorization header will vary from request to request. If the request signature calculated by the system matches the signature included with the request, the requester will have demonstrated possession of the NFON secret access key. The request will then be processed under the identity, and with the authority, of the developer to whom the key was issued.

Following is pseudogrammar that illustrates the construction of the Authorization request header. (In the example, \n means the Unicode code point U+000A, commonly called newline). **A real-world implementation based on JavaScript for that can be found in chapter 10.**

```
Authorization = "NFON-API" + " " + NFONAccessKeyId + ":" + Signature;

Signature = Base64 (HMAC-SHA1(YourSecretAccessKeyID, UTF-8-Encoding-Of(StringToSign) ) );

StringToSign =

     HTTP-Verb + "\n" +
     Content-MD5 + "\n" + Content-Type + "\n" + Date + "\n" +
     CanonicalizedNfonHeaders + CanonicalizedResource;

CanonicalizedResource = [Absolute Path];

CanonicalizedNfonHeaders = <described below>
```

HMAC-SHA1 is an algorithm defined by RFC 2104 - Keyed-Hashing for Message Authentication. The algorithm takes as input two byte-strings, a key and a message. For NFON API request authentication, use your NFON secret access key (YourSecretAccessKeyID) as the key, and the UTF-8 encoding of the StringToSign as the message. The output of HMAC-SHA1 is also a byte string, called the digest. The Signature request parameter is constructed by Base64 encoding this digest.

Content-Type cannot be empty. Content-MD5, one must use the MD5 of the body even if the body is empty.

4. Request Canonicalization for Signing

Recall that when the system receives an authenticated request, it compares the computed request signature with the signature provided in the request in **StringToSign.** For that reason, you must compute the signature by using the same method used by NFON We call the process of putting a request in an agreed-upon form for signing **canonicalization.**

5. Constructing the Canonicalized NFON Headers Element

To construct the CanonicalizedNfonHeaders part of **StringToSign,** select all HTTP request headers that start with 'x-nfon-' (using a case-insensitive comparison), and use the following process.

CanonicalizedNfonHeaders Process

- Convert each HTTP header name to lowercase. For example, 'X-NFON-Date' becomes 'x-nfon- date'.
- Sort the collection of headers lexicographically by header name.
- Combine header fields with the same name into one "header-name:comma-separated-value- list" pair as prescribed by RFC 2616, section 4.2, without any whitespace between values. For example, the two metadata headers 'x-nfon-meta-username: fred' and 'x-nfon-meta-username: barney' would be combined into the single header 'x-nfon-meta-username: fred,barney'.
- "Unfold" long headers that span multiple lines (as allowed by RFC 2616, section 4.2) by replacing the folding whitespace (including new-line) by a single space.
- Trim any whitespace around the colon in the header. For example, the header 'x-nfon-meta- username: fred,barney' would become 'x-nfon-meta-username:fred,barney'
- Finally, append a newline character (U+000A) to each canonicalized header in the resulting list. Construct the CanonicalizedResource element by concatenating all headers in this list into a single string.

6. Positional versus Named HTTP Header StringToSign Elements

The first few header elements of **StringToSign** (Content-Type, Date, and Content-MD5) are positional in nature. **StringToSign** does not include the names of these headers, only their values from the request. In contrast, the 'x-nfon-' elements are named. Both the header names and the header values appear in **StringToSign.**

If a positional header called for in the definition of **StringToSign** is not present in your request (for example, Content-Type or Content-MD5 are optional for PUT requests and meaningless for GET requests), substitute the empty string ("") for that position.

7. Time Stamp Requirement

A valid time stamp (using either the HTTP Date header or an x-nfon-date alternative) is mandatory for authenticated requests. Furthermore, the client timestamp included with an authenticated request must be within 15 minutes of the NFON system time when the request is received. If not, the request will fail with the *RequestTimeTooSkewed* error code. The intention of these restrictions is to limit the possibility that intercepted requests could be replayed by an adversary. For stronger protection against eavesdropping, use the HTTPS transport for authenticated requests.

The validation constraint on request date applies only to authenticated requests that do not use query string authentication. For more information, see Query String Request Authentication Alternative.

Some HTTP client libraries do not expose the ability to set the Date header for a request. If you have trouble including the value of the 'Date' header in the canonicalized headers, you can set the timestamp for the request by using an 'x-nfon-date' header instead. The value of the x-anfon-

date header must be in one of the RFC 2616 formats (www.ietf.org/rfc/rfc2616.txt). When an x-nfon- date header is present in a request, the system will ignore any Date header when computing the request signature. Therefore, if you include the x-nfon-date header, use the empty string for

the Date when constructing the *StringToSign.* See the next section for an example.

8. Authentication Examples

The examples in this section use the (non-working) credentials in the following table:

Parameter	Value
NfonAccessKeyId	EXAMPLE-02ED-4E2B-AC15-01F8C92E2D86
NfonSecretAccessKey	EXAMPLE-E5DD-4AC1-99DB-23FFB50A18F6

In the example **StringToSign**, formatting is not significant, and \n means the Unicode code point U+000A, commonly called newline.

Example Object GET

Request	StringToSign
GET /api/customers/K4076/targets/ HTTP/1.1 Accept : */* Content-MD5: Content-Type : Authorization : NFON-API EXAMPLE-02ED-4E2B-AC15-01F8C92E2D86:f6g/fdpcGeA7EGn2m5LmeKLDGNw= Host : portal-api.nfon.net	GET Wed, 29 Nov 2023 15:20:18 GMT /api/customers/K4076/targets/phone-extensions/1404 → Signature f6g/fdpcGeA7EGn2m5LmeKLDGNw=

Example Object PUT

Request	StringToSign

PUT /api/customers/K4076/targets/phone-extensions/1404 HTTP/1.1
Accept : */*
Content-MD5 : 45d08d9b6f2d2fe940399b2bfdaeb7df
Host : portal-api.nfon.net
Content-Type : application/json
Authorization : NFON-API EXAMPLE-02ED-4E2B-AC15-01F8C92E2D86:PfHTbQHTaK8tA4b5FTNNaiFOv/I=
x-nfon-date : Wed, 29 Nov 2023 18:02:09 GMT
 	
Content-Length : 124
{
  "data": [
      {
          "name": "displayName",
          "value": "NFON Extension changed"
      }
  ]
}
	PUT
	45d08d9b6f2d2fe940399b2bfdaeb7df
	application/json
	Wed, 29 Nov 2023 18:02:09 GMT
	/api/customers/K4076/targets/phone-extensions/1404
	
	→ Signature
	PfHTbQHTaK8tA4b5FTNNaiFOv/I=

Note the Content-Type header is required when the request body is not empty like in this case.
The Content-Type header in the request and in the StringToSign. Also note that the Content-MD5 is left blank in the StringToSign, because it is not present in the request.

Example Object DELETE

This example deletes a phone extension.

Request	StringToSign
DELETE /api/customers/K4076/targets/phone-extensions/1404 HTTP/1.1 Host : portal-api.nfon.net Content-Type : x-nfon-date : Wed, 29 Nov 2023 15:43:18 GMT Accept : */* Content-MD5 : Authorization : NFON-API EXAMPLE-02ED-4E2B-AC15-01F8C92E2D86:lm4p8HIC0k1qhtgOplkGU6JMSCo=	 DELETE Wed, 29 Nov 2023 15:48:38 GMT /api/customers/K4076/targets/phone-extensions/1404 → Signature lm4p8HIC0k1qhtgOplkGU6JMSCo=

Note how we used the alternate 'x-nfon-date' method of specifying the date (because our client library prevented us from setting the date, say). In this case, the x-nfon-date takes precedence over the Date header. Therefore, date entry in the signature must contain the value of the x-nfon-date header.

Example POST

Request	StringToSign

POST /api/customers/K4076/targets/phone-extensions HTTP/1.1
Accept : */*
Authorization : NFON-API EXAMPLE-02ED-4E2B-AC15-01F8C92E2D86:uEdHxB623HK/sSY0GsNCSwIf2XA=
Content-Length : 46
x-nfon-date : Mon, 16 Mar 2020 10:11:14 GMT Host : portal-api.nfon.net
Content-Type : application/json
Content-MD5 : 5d6651206f3687d15d4324326a59d367
{
    "data": [
        {
            "name": "extensionNumber",
            "value": "1404"
        },
        {
            "name": "displayName",
            "value": "My Extension"
        },
        {
            "name": "accessCentralPhoneBook",
            "value": true
        },
        {
            "name": "autodialTimeout",
            "value": 0
        },
        {
            "name": "intercomEnabled",
            "value": false
        },
        {
            "name": "numberguessingLength",
            "value": 0
        },
        {
            "name": "callWaitingIndication",
            "value": true
        }
    ]
}
	
	POST
	5d6651206f3687d15d4324326a59d367
	application/json
	Mon, 16 Mar 2020 10:11:14 GMT
	/api/demo
	
	POST
	27c1474221842e5e49f65791c6345128
	application/json
	Wed, 29 Nov 2023 17:42:25 GMT
	/api/customers/K4076/targets/phone-extensions
	
	→ Signature
	uEdHxB623HK/sSY0GsNCSwIf2XA==

Note how we used the alternate 'x-nfon-date' method of specifying the date. In this case, the x-nfon-date takes precedence over the Date header. Therefore, date entry in the signature must contain the value of the x-nfon-date header.

Special Case PUT and POST with audio files

In the special case where data is sent together with audiofiles this needs to be treated identical to a GET or DELETE request *without* a body where the contentMD5 and the content-type is *not* added to the computation of the signature.

An example for this would be the call to /api/customers/<cid>/announcements/<aid> where the data has to be provided as well as the file itself.

Hint: have a close look at the API endpoint documentation to make sure the values and mime-types are correctly set as this can easily lead to wrong request or authentication failures.

See chapter 10 for an example of detecting this situation.

Example Upload

Request	StringToSign

`PUT /photos/puppy HTTP/1.1 User-Agent: curl/7.15.5`
`Host: static.johnsmith.net:8080 Date: Tue, 27 Mar 2007`
`21:06:08`
`+0000`
`x-nfon-acl: public-read content-type: application/x-`
`download`
`Content-MD5: 4gJE4saaMU4BqNR0kLY+lw== X-NFON-Meta-`
`ReviewedBy: joe@johnsmith.net`
`X-Nfon-Meta-ReviewedBy: jane@johnsmith.net`
`X-NFON-Meta-FileChecksum: 0x02661779`
`X-Nfon-Meta-ChecksumAlgorithm: crc32`
`Content-Disposition: attachment; filename=database.dat`
`Content-Encoding: gzip Content-Length: 5913339`
`Authorization: NFON-API AKIAIOSFODNN7EXAMPLE:`
`ilyl83RwaSoYIEdixDQcA4OnAnc=`
	`PUT\n`
	`4gJE4saaMU4BqNR0kLY+lw==\n`
	`application/x-download\n`
	`Tue, 27 Mar 2007 21:06:08 +0000\n`
	`x-nfon-acl:public-read\n`
	`x-nfon-meta-checksumalgorithm:crc32\n`
	`x-nfon-meta-filechecksum:0x02661779\n`
	`x-nfon-meta-reviewedby:`
	`joe@johnsmith.net,jane@johnsmith.net\n`
	`/photos/puppy`

Notice how the 'x-amz-' headers are sorted, trimmed of whitespace, and converted to lowercase. Note also that multiple headers with the same name have been joined using commas to separate values. Note how only the Content-Type and Content-MD5 HTTP entity headers appear in the StringToSign. The other Content-* entity headers do not. Again, note that the CanonicalizedResource includes the bucket name, but the HTTP Request-URI does not. (The bucket is specified by the Host header.)

9. REST Request Signing Problems

When REST request authentication fails, the system responds to the request with an XML error document. The information contained in this error document is meant to help developers diagnose the problem. In particular, the StringToSign element of the SignatureDoesNotMatch error document tells you exactly what request canonicalization the system is using.

Some toolkits silently insert headers that you do not know about beforehand, such as adding the header Content-Type during a PUT. In most of these cases, the value of the inserted header remains constant, allowing you to discover the missing headers by using tools such as Ethereal or tcpmon.

Using Base64 Encoding

HMAC request signatures must be Base64 encoded. Base64 encoding converts the signature into a simple ASCII string that can be attached to the request. Characters that could appear in the signature string like plus (+), forward slash (/), and equals (=) must be encoded if used in a URI. For example, if the authentication code includes a plus (+) sign, encode it as %2B in the request. Encode a forward slash as %2F and equals as %3D.

NFON **AG**

10. Postman Pre-Request-Script

The following script resembles code that can be injected into the pre-request-method of Postman. It can be used to see exactly how the signing of the request can be implemented:

```
/*
 * Postman Pre-Request-Script to sign an API-Request with a provided API-key
 * copyright NFON AG 2024
 *
 * 15/03/2024 Fix: Params need to be encoded before signing
 *
 * This script should be added to the pre-script section of a collection
 * For convenience the script even allows to use {{variables}} in the URL as well as within the body
 *
 * The following variables should be defined in the environment
 * - apiKey : Set to the api or access key provided by NFON
 * - apiSecret : Set to the api or access secred provided by NFON
 * - contentMD5
 * - nfonTimestamp
 * - apiSignature
 * - baseUrl: Set to "https://portal-api.nfon.net:8090" (e.g, for using it in the URL itself
 *             {{baseUrl}}/api/customers/K4076/targets/phone-extensions/1404
 *
 * Setup the headers as follows
 *
 * Accept: * / *
 * Authorization: NFON-API {{apiKey}}:{{apiSignature}}
 * Content-MD5: {{contentMD5}}
 * Content-Type: application/json
 * x-nfon-date: {{nfonTimestamp}}
 */

postman.setEnvironmentVariable("contentMD5", '');
postman.setEnvironmentVariable("nfonTimestamp", '');
postman.setEnvironmentVariable("apiSignature", '');

 /*
  * Signing function
  * - Takes all parameters (except the apikey itself)
  * - Then concatenates them together with CRLF to one string
  * - Use HMACSHA1 by applying the apiSecret
  * - Finally base64 encode the result
  */
var sign = function() {
    var args = [].slice.call(arguments);

   console.log("Signing Request")
   console.log("apiSecret="+args[0])
   console.log("method="+args[1])
   console.log("contentMD5="+args[2])
   console.log("content-type="+args[3])
   console.log("date="+args[4])
   console.log("url="+args[5])

   const url = args[5]
   const [baseUrl, queryString] = url.split("?");

   if (queryString !== undefined) {
       let encodedQueryString = ""
       const params = queryString.split("&");

       const encodedParams = params.map(param => {
       const [key, value] = param.split("=");
       return `${key}=${encodeURIComponent(value)}`;
       });

       encodedQueryString = encodedParams.join("&");
       args[5] = `${baseUrl}?${encodedQueryString}`
   }

   console.log("endcoded url="+args[5]);

   var passphrase = args.splice(0, 1);

   //removing apiSecret from args
   var arrayLength = args.length;
   var message = '';

   // add CRLF to each of the arguments and concatenate them to one string
   for (var i = 0; i < arrayLength; i++) {
       if (args[i] !== '') {
           message = message + args[i] + '\n';
       }
   }

   // remove last CRLF
   message = message.substring(0, message.length - 1);

   console.log("concatenated arguments as message: >\n" + message+"\n<");
   console.log("using apikey to encrypt message: " + passphrase.toString() + "\n")
```

```
    console.log("HMACSHA1:\n"+CryptoJS.HmacSHA1(message, passphrase.toString()))
    // Use HMAC with SHA1 to encrypt the message with the apikay and then convert to Base64
    // see https://en.wikipedia.org/wiki/HMAC
    return CryptoJS.HmacSHA1(message, passphrase.toString()).toString(CryptoJS.enc.Base64);
};

 // Start of the Pre-Request Processing

 if (!environment.apiSecret) {
     // Show log with "view->console"
     console.log('Error: API Secret not set');
     return;
 }

 var date = new Date().toUTCString();

 // at this time, the variables in the URL have not been resolved yet
 var Property = require('postman-collection').Property;

 // remove {{baseUrl}} and extract url

 var url = request.url.trim().split("{{baseUrl}}");
 var route = url[1];

 console.log("original url path " + route);
 console.log("postman variables " + JSON.stringify(pm.variables));

 // at this time, variables in the URL are not replaced yet; perform substitution for signature calculation
 var routeWithResolvedVariables = Property.replaceSubstitutions(route, pm.variables.toObject())

 console.info("final url path" + routeWithResolvedVariables);
 console.info("request.method " + request.method);
 console.log("request " + JSON.stringify(request.data));

 // the following block checks if an audiofile is provided in the body
 // by checking if a property "audiofile" is present

 var hasAudioFile = false;
 if (request.data && (request.method === 'GET' || request.method === 'PUT' || request.method === 'POST')) {
    var requestString = JSON.stringify(request.data)
    var requestBodyString = JSON.stringify(request.data.audiofile1)
    console.log("original request data" + requestString)
     var theRequest = Property.replaceSubstitutions(request.data.toString(), pm.variables.toObject())
     console.log("request data with replaced variables" + request.toString())
     var mydata = JSON.stringify(theRequest)
     console.log("request data BEFORE replacing variables: json" + mydata)
     mydata = mydata.replace(/\\n/g, "")
                    .replace(/\\r/g, "")
                     .replace(/\\\"/g, "\"")
                     .replace(/\"{/g, "{")
                     .replace(/}\"/g, "}");
     console.log("request data with replaced variables: json" + mydata.toString())
     dataObj = JSON.parse(mydata)
     hasAudioFile = request.data.audiofile1 === undefined
     console.log("hasAudioFile: "+ hasAudioFile)

 }


 // check if we either have GET without body or sending an audiofile
 // Delete request always leave out body for the signature
 if (request.method === 'DELETE' || hasAudioFile || (typeof request.data == "object" && _.isEmpty(request.data) && request.method ===
'GET')) {
     console.info(">>>GET without Body data<<<")
     // On a GET request without a body we do not take the contendMD5 and content-type into account of what is signed
     var signature = sign(environment.apiSecret, request.method, '', '', date, routeWithResolvedVariables);
     // make sure contentMD5 is not set for request
     postman.setEnvironmentVariable("contentMD5", "");
 } else {
     console.info(">>>ANY Request with Body data<<<")
     // at this time, variables in the body are not replaced yet; perform substitution for MD5 / signature calculation
     var bodyWithResolvedVariables = Property.replaceSubstitutions(request.data.toString(), pm.variables.toObject())

     var contentMD5 = (request.method != 'GET' && request.method != 'DELETE')? CryptoJS.MD5(bodyWithResolvedVariables.toString()) :
CryptoJS.MD5('');
     var signature = sign(environment.apiSecret, request.method, contentMD5, request.headers['content-type'], date,
routeWithResolvedVariables);

     console.info("contentMD5: " + contentMD5)
     postman.setEnvironmentVariable("contentMD5", String(contentMD5));
 }
 console.info ("nfonTimestamp: " + date)
 postman.setEnvironmentVariable("nfonTimestamp", date);
 console.info ("apiSignature: " + signature)
 postman.setEnvironmentVariable("apiSignature", signature);

 console.info("+++ Finished header calculation +++");
```

11. API Endpoints

Please find the API endpoints documentation here.

Download