Application Layer REST-API Documentation

Ceyoniq

Version 10.1.1501, 05.03.2026

Table of Contents

Overview

This document describes the REST-API and resources provided by the Application Layer. The REST API's are for developers who want to integrate the Application Layer into their application and for administrators who want to script interactions with the Application Layer Server.

The Application Layer's REST API's provide access to resources via URI paths. To use a REST API, your application will make an HTTP request and parse the response. The response format is JSON or XML. Your methods will be the standard HTTP methods like GET, PUT, POST and DELETE.

Because the REST API is based on open standards, you can use any web development language to access the API.

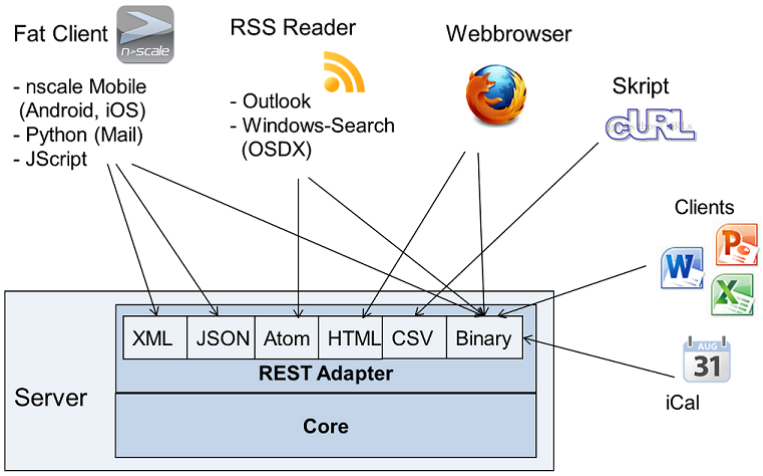

Figure 1. Clients for REST interface

Chapter 1. Introduction

Accessing REST API is performed via HTTP. When you enter a URL into a web browser address bar, the browser performs an HTTP GET request to the URL. This usually returns a web page in the form of an HTTP response that the browser displays. But the GET method is one of several HTTP request methods. The Application Layer REST API uses the four main HTTP methods: GET, POST, PUT, and DELETE. The most widespread methods are GET and POST. The other methods are less known but they became widely known due to the popularity of REST web services. An important concept of the REST architecture is that different HTTP request methods perform different actions when applied to the same URL.

Verb	Description
`GET`	Read a resource, `Accept` header forces representation
`HEAD`	Read a resource, like `GET`, but only HTTP header are returned, the body is skipped
`POST`	Create a new resource, result contains the URI of the created resource
`DELETE`	Delete a resource
`PUT`	Modify a resource for the given URI (create a new resource)

The Application-Layer REST-API supports the following representation formats:

Programming representations

- XML (application/xml)
- JSON (application/json)

Other representations

- XHTML / HTML (application/html+xml)
- Text (text/plain, text/csv)
- Binary (PDF, Office, etc.)

Chapter 2. REST URI's, Header and Parameter

The chapter describes the basic HTTP resources of the Application Layer.

2.1. Service URLs

URI's for a Application-Layer REST API resource have the following structure:

`http(s)://hostname:port/nscalealinst1/rest/service/resource-name`

The Application Layer only uses a self-signed certificate by default. You can replace this server certificate with a digitally signed certificate by a certificate authority (CA).

The self-signed certificate can be downloaded at `http(s)://hostname:port/server.certificate`

Table 1. Port

Number	Description
8080	plain HTTP port
8443	secure HTTPS port

The logical application layer instance is part of the URL. Your application should make this entry configurable.

Table 2. Instance

Name	Description
nscalealinst1	The default application-layer instance name (can be changed)
[any-name]	Secondary logical instance name

The service name part must not be configurable. Here are the available services:

Table 3. Core Services

Name	Description
authoritymanagement	authority management functionality: roles
configuration	configuration functionality: dictionary, layouts, property definitions, value sets
masterdata	masterdata functionality: external data
monitoring	monitoring functionality: monitoring data, invoke generic
messaging	messaging functionality: subscribe resources and workflow, read messages
repository	repository functionality: folder, link and document management

Name	Description
usermanagement	user management functionality: principals, org. entities, groups and users.
workflow	workflow functionality: processes and taks

Table 4. Resources

Name	Description
/docares/{name}	document area name as a parameter
/doc/{id}	document identifier as a parameter
/folder/{id}	folder identifier as a parameter

This are only some examples of a resource path. A complete reference is available at:

`http(s)://hostname:port/index.html`

The HTML document `rest-api.html` contains the full resource listing.

2.2. Common Request Header and Query Parameter

The following table describes headers that can be used by various types of REST requests.

Table 5. Request Header

Header	Description
Authorization	The information required for request authentication
Accept	Media type that is acceptable for the response (content negotiation).
Content-Type	Media type of the body of the request (used with POST and PUT requests).

Table 6. Query Parameter

Header	Description
appid	The client application id of the client. If no appid is given, the client requires the 'nscale SDK' license.
autoclose=true	The application cannot use the session cookie. Close session after request.
clientversion	The client application version.
properties	A list of nscale system of user defined properties for a GET requests.

2.3. Common Response Header

The following table describes response headers that are common to most REST responses.

Table 7. Response Header

Header	Description
Content-Length	Length of the message (without the headers) according to RFC 2616
Content-Type	The content type of the resource in case the request content in the body
Location	The URI of the created resource (by PUT or POST request)

Chapter 3. REST Examples

First try to access the application layer via the cURL command line tool.

Here is an example session:

```
curl  -v --basic -u admin:admin
      -H "Accept: application/json"
      http://localhost:8080/nscalealinst1/rest/repository/docarea/DA/root

GET /nscalealinst1/rest/repository/folder/DA$NOTSET$-1$1$NOTSET/children.json HTTP/1.1
Authorization: Basic YWRtaW46YWRtaW4=
User-Agent: curl/7.21.1 (i386-pc-win32) libcurl/7.21.1 zlib/1.2.5
Host: localhost:8080
Accept: application/json
```

Answer from Application Layer:

```
HTTP/1.1 200
Set-Cookie: JSESSIONID=C829DE8E7C22CA3BF728A65EC880D135;path=/nscalealinst1;HttpOnly
Content-Type: application/json
Content-Length: 1194
Date: Tue, 17 Jan 2017 14:09:03 GMT

<< body >>
```

The same request with java code. The code uses the JAX-RS 2.0 specification.

```
static String getRoot ( String areaname ) {

    String restUrl = "http://localhost:8080/nscalealinst1/rest";
    URI uri = new URI( restUrl + "/repository/docarea/" + areaname + "/root" );

    String auth = "Basic " + new String(Base64.encodeBase64("admin:admin".getBytes()))

    Client client = ClientBuilder.newClient();

    try {
        return client.target(uri).request()
                     .header ( "Authorization", auth )
                     .accept ( MediaType.APPLICATION_JSON )
                     .get ( String.class );
    } finally {
       client.close();
    }
}
```

The same request with C# code. The code uses the HttpClient object (requires Framework 4.5).

```
static async Task < String > getRoot()
{
    using (var httpClient = new HttpClient())
    {
        httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue( "Basic",
        Convert.ToBase64String( System.Text.ASCIIEncoding.ASCII.GetBytes(
                                string.Format("{0}:{1}", "admin", "admin"))));

        var response = await httpClient.GetAsync(baseAddress +
"/repository/docarea/DA/root.json");
        return await response.Content.ReadAsStringAsync();
    }
}
```

Programming

This chapter describes details for programming.

Chapter 4. REST Client

Lets write a typical client for the REST-API.

4.1. Login

There is no special REST URL to send the credentials. The standard HTTP authentication mechanism is used by REST:

- Basic
- NTLM
- Negotiate (Kerberos or NTMLv2)
- OpenID Connect (ADFS)

Additional authentication schemas:

- Implicit (can be used for impersonation)
- AuthID (ID/secure-card based login)
- KNM (Kyocera Network Manager)
- SAML (planned)

The server uses a session cookie. The client should use this cookie for following requests.

Resource name: `/rest/login`

This resource retrieves additional information of the session access rights.

Note: if the client cannot use cookies, use the autoclose query parameter. Example: `/rest/resource?autoclose`

4.2. Logout

Resource name: `/rest/logout`

This resource will close the server session and the HTTP session for the REST adapter. You can also use the query parameter `autoclose` to force the logout after one request.

Caution: just disconnecting the HTTP connection will leave an open session in the server.

4.3. Document areas

The client should allow the user to select a document area. Therefore you should detect the possible choices.

Get all available document areas for the current user:

```
curl  -v --basic -u admin:admin
      -H "Accept: application/json"
      http://localhost:8080/nscalealinst1/rest/repository/docareas

> GET /nscalealinst1/rest/repository/docareas HTTP/1.1
> Authorization: Basic YWRtaW46YWRtaW4=
> User-Agent: curl/7.54.0
> Host: localhost:8080
> Accept: application/json
```

The HTTP header "Accept" controls the representation of the response body.

Answer from Application Layer:

```
< HTTP/1.1 200
< Set-Cookie: JSESSIONID=46312ED0448E60152813EE5E8CECD849;path=/nscalealinst1;HttpOnly
< Content-Type: application/json
< Content-Length: 644
< Date: Thu, 19 Jan 2017 15:43:13 GMT
```

```
{"docAreas":[
  { "areaName": "DA",
    "displayNameId": "DA",
    "rootFolderId": "DA$NOTSET$-1$1$NOTSET",
    "personalFolderId": "DA$NOTSET$60$1$NOTSET"
  },
  { "areaName": "APPS", "displayNameId": "APPS",",  ... }
]}
```

The current user can access the document areas `DA` and `APPS`. There are two starting points for the repository. The `rootFolderId` is the global repository key and `personalFolderId` is the personal folder repository id.

4.4. Repository

4.4.1. Get the root folder from a document area:

To start on a document area you must retrieve the root entry point (resource key).

```
curl  -v --basic -u admin:admin
      -H "Accept: application/json"
      http://localhost:8080/nscalealinst1/rest/repository/docarea/DA/root

> GET /nscalealinst1/rest/repository/docareas HTTP/1.1
> Authorization: Basic YWRtaW46YWRtaW4=
> User-Agent: curl/7.54.0
```

```
> Host: localhost:8080
> Accept: application/json
```

The HTTP header "Accept" controls the representation of the response body.

Answer from Application Layer:

```
< HTTP/1.1 200
< Set-Cookie: JSESSIONID=46312ED0448E60152813EE5E8CECD849;path=/nscalealinst1;HttpOnly
< Content-Type: application/json
< Content-Length: 1133
< Date: Thu, 19 Jan 2017 15:43:13 GMT
```

```
{ "resourceKey":
   { "id":"DA$NOTSET$-1$1$NOTSET",
     "areaName":"DA",
     "type":"FOLDER"},
  "properties":[ ... ]
}
```

4.4.2. Get the children of a folder element:

To display the child elements of a folder, you must retrieve the possible children.

```
curl  -v --basic -u admin:admin
      -H "Accept: application/json"
      http://localhost:8080/nscalealinst1/rest/repository/folder/DA$NOTSET$-
1$1$NOTSET/children?properties=displayname

> GET /nscalealinst1/rest/repository/docareas HTTP/1.1
> Authorization: Basic YWRtaW46YWRtaW4=
> User-Agent: curl/7.54.0
> Host: localhost:8080
> Accept: application/json
```

Note that the query parameter "properties" defines the returned property values.

Answer from Application Layer:

```
< HTTP/1.1 200
< Set-Cookie: JSESSIONID=46312ED0448E60152813EE5E8CECD849;path=/nscalealinst1;HttpOnly
< Content-Type: application/json
< Content-Length: 1198
< Date: Thu, 19 Jan 2017 15:43:13 GMT
```

```
{  "id":"DA$NOTSET$-1$1$NOTSET",
```

```
   "displayName":"DA",
   "count":1,
   "items": [ { "resourceKey":{"id":"DA$NOTSET$103$1$NOTSET","areaName":"DA","type":
"FOLDER"},
                "properties": [{"name":"displayname","value":"Rainer"} ]}
            ]
}
```

The result is a structure with resource keys and their properties.

4.4.3. Search for elements:

The search query contains the NQL (nscale query language) string for searching. Please read the NQL reference document.

```
curl  -v --basic -u admin:admin
      -H "Accept: application/json"
      "http://localhost:8080/nscalealinst1/rest/repository/folder/DA$NOTSET$-
1$1$NOTSET/search?query=select%20displayname"

> GET /nscalealinst1/rest/repository/docareas HTTP/1.1
> Authorization: Basic YWRtaW46YWRtaW4=
> User-Agent: curl/7.54.0
> Host: localhost:8080
> Accept: application/json
```

The HTTP header "Accept" controls the representation of the response body.

Answer from Application Layer:

```
< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Set-Cookie: JSESSIONID=BDBDBD64C26F8E7511AF318C4EEB8991; Path=/nscalealinst1;
HttpOnly
< Content-Type: application/json
< Content-Length: 208
< Date: Tue, 13 Jun 2017 08:59:23 GMT
```

```
{
   "id":"DA$NOTSET$-1$1$NOTSET",
   "displayName":"SEARCH",
   "count":1,
   "items":[ {"resourceKey":{"id":"DA$NOTSET$103$1$NOTSET","areaName":"DA","type":
"FOLDER"},
              "properties":[{"name":"displayname","value":"Rainer"}]}
           ]
}
```

The result is a structure with resource keys and their properties.

4.4.4. Modify properties:

The request is uses the PUT method (modify). For the application layer it behaves like a PATCH request. Only the given properties are changed.

```
curl  -v --basic -u admin:admin
      -H "Accept: application/json"
      -H "Content-Type: application/json"
      -X PUT
      --data-binary @file.json
      http://localhost:8080/nscalealinst1/rest/repository/doc/test$NOTSET$53$2$NOTSET

> PUT /nscalealinst1/rest/repository/folder/DA$NOTSET$153$1$NOTSET HTTP/1.1
> Host: localhost:8080
> Authorization: Basic YWRtaW46YWRtaW4=
> User-Agent: curl/7.54.0
> Accept: application/json
> Content-Type: application/json
> Content-Length: 126
```

JSON document (file.json)

```
{
  "properties": [
      {
           "name": "displayname",
           "value": "Value of Displayname"

           "name": "latitude",
           "value": {
               "list": [ 1.0, 2.0 ]
           }
      }
   ]
}
```

Answer from Application Layer:

```
< HTTP/1.1 204 No Content
< Server: Apache-Coyote/1.1
< Set-Cookie: JSESSIONID=8FFEAC201490545C81E33B0E2FFB468E; Path=/nscalealinst1;
HttpOnly
< Date: Tue, 13 Jun 2017 08:56:16 GMT
<
```

The HTTP code 204 means that nothing is returned here.

4.4.5. Document upload

A document upload should be an multipart/mixed request. The first part contains the metadata and the next parts one or more payload's.

The first part must have the media type "application/json" or "application/xml".

```
curl  -v --basic -u admin:admin
      -H "Accept: application/json"
      -H "Content-Type: multipart/mixed"
      -F "metadata=@metadata.json; type=application/json"
      -F "content=@MyFileToUpload.jpg; type=image/jpg"
      http://localhost:8080/nscalealinst1/rest/repository/doc/test$NOTSET$53$2$NOTSET

> GET /nscalealinst1/rest/repository/docareas HTTP/1.1
> Authorization: Basic YWRtaW46YWRtaW4=
> User-Agent: curl/7.54.0
> Host: localhost:8080
> Accept: application/json
```

JSON document (metadata.json)

```
{
  "properties":[{
    "name":"displayname",
    "value":"The displayname value"
  }],
  "objectclassname":"D1",
  "contentItemProperties":[{
    "properties":[ {
      "name":"name",         "value":"The item name value" }, {
      "name":"displayName",  "value":"The item displayname value" } , {
      "name":"contentType",  "value":"image/jpeg" }, {
      "name":"lastModified", "value":1393875885230
      } ]
   }],
  "contentProperties":[{
     "name":"contentType",    "value":"text/plain; charset=\"iso-8859-1\""
   }]
}
```

The curl parameter "-F" must be used for multipart documents.

Answer from Application Layer:

```
< HTTP/1.1 201 Created
< Server: Apache-Coyote/1.1
< Set-Cookie: JSESSIONID=F09888110FD98E2C98FA4E44249870A9; Path=/nscalealinst1;
HttpOnly
```

```
< Location:
http://localhost:8080/nscalealinst1/rest/repository/doc/DA$NOTSET$364$2$NOTSET
< Content-Type: application/json
< Content-Length: 65
< Date: Tue, 13 Jun 2017 09:17:42 GMT
```

```
{  "id":"DA$NOTSET$364$2$NOTSET",
   "areaName":"DA",
   "type":"DOCUMENT"
}
```

The location header contains the new resource URL. The body contains the resource key element.