NOKIA

Network Services Platform Release 2.0 R4

API Programmer Guide

3HE-11082-AAAD-TQZZA

Issue 1

December 2016

Legal notice

Contents

About this document

Purpose

The *NSP API Programmer Guide* provides information to assist with the creation of scripts that interact with the NSP in order to perform various functions.

Document support

Customer documentation and product support URLs:

Customer documentation welcome page

- https://infoproducts.alcatel-lucent.com/cgi-bin/doc_welc.pl

Technical support

- http://support.alcatel-lucent.com

How to comment

Documentation feedback

- Documentation Feedback

1 Getting Started

1.1 Introduction

1.1.1 Before you begin

Nokia recommends you do the following before you start using the NSP API:

- review the license requirements for your NSP system
- understand the following technologies, which are beyond the scope of this guide:
  - HTTP(S): *HyperText Transfer Protocol (Secure)*
  - REST architecture: *Representational State Transfer*
  - JSON: *Javascript Object Notation*
  - web applications

1.1.2 NSP web applications

The NSP provides service creation and network optimization functions using browser-based web applications. Each of these applications can be accessed through a browser by any authorized user. To view a dashboard of all the available applications, go to:

https://<*server*>:8543/webLaunchpad

Where *server* is the hostname or IP address of your installed NSP server.

The HTTP(S) protocol is used between the OSS application and the NSP applications. All standard HTTP error codes must be handled accordingly.

1.2 NSP REST API design

1.2.1 General information

The NSP and its applications provide dynamic connectivity and resource control using high-level, RESTful APIs. The REST requests are formatted using UTF-8 in JSON.

To view and interact with the APIs online, go to:

- https://<*server*>:8543/sdn/doc
- https://<*server*>:8543/task-scheduler/doc
- https://<*server*>:8543/ean/doc

Where *server* is the hostname or IP address of your installed NSP server.

From these sites, users can browse categorized API methods, view detailed request and response models, and send requests with minimal configuration.

Note: Offline representations of the API methods and models are provided alongside the full suite of NSP user documentation.

1.2.2 Versioning

Up to three versions of the REST APIs are maintained per major release. The latest API version may introduce additional request attributes if there is a fully backward-compatible default value, and users should be prepared to either accept additional response attributes, or ignore any that are unknown. The latest API versions may also introduce additional functions, but the behavior of existing functions will not change.

Note: For a detailed summary of the differences between API versions, see the *NSP API Differences Guide*.

1.2.3 Command response object

The NSP REST APIs are designed to support pagination by a JSON object that includes the values “startRow”, “endRow”, “totalRows”, and “status”. The APIs described in this document do not use pagination, as the NSP server always returns a complete response. Response filtering in the request is not available in the current API.

1.2.4 HTTP headers

The NSP REST API request uses HTTP headers for authentication. HTTP headers are used instead of query parameters for several reasons, mostly because it keeps the resource URI clean, short and easy to read.

Content-type

The server response will be formatted in JSON. Content-Type is a required header field for HTTP POST, PUT, and PATCH as mentioned below to either create or modify an entity:

PUT /whatever HTTP/1.1

Content-Type: application/json

Authorization: $AUTHORIZATION_STRING

Authorization (required)

Each API call must be authenticated, so the Authorization header must be sent with each request (with the exception of the GET token request):

GET /whatever HTTP/1.1

Authorization: $AUTHORIZATION_STRING

$AUTHORIZATION_STRING is the token returned by the GET token request.

1.2.5 Server response codes

The server replies using standard HTTP response codes.

2XX codes

2XX codes indicate that the request was processed
- 200: Everything is OK (content is in the body)
- 204: Everything is OK, but there is no content in the body (this is the standard response for an update or a deletion)

4XX codes
- 400: Bad request: something is wrong in the request
- 401: Not authenticated (wrong password or wrong Token)
- 403: Not authorized (request for something to which you do not have rights)
- 404: Not found (request for something that does not exist)
- 409: Conflict (the requested change conflicts with existing configuration)

5XX codes
- 500: Internal server error (error is explained in the body)

1.2.6 Further API concepts

Knowledge of the following concepts will facilitate usage of the NSP API.

Using Application IDs

The Application ID property is a highly searchable field that allows for integration with preexisting infrastructure, or other systems that use object IDs. When Application IDs are used, correlation of objects can be performed quickly and without the need of full discovery.

To set an Application ID on an object, you must either configure the *appId* property at the time of its creation, or use the generic/application-id/{id}/{appId} REST call to modify it later. It is recommended that these IDs be prefixed with an application-specific prefix.

Custom attributes

Custom attributes can be used to extend the NSD models and change the standard provisioning behavior towards devices. In create or modify requests for services or

endpoints, the custom attributes can be provided as attributeName/attributeValue string lists. The NSD then passes these parameters to the device mediation layer, which can use them when configuring services and service endpoints. Custom attributes are supported for mediation by 5620 SAM.

In a default NSP configuration, custom attributes can reference the 5620 SAM object model for services and endpoints. This allows a client of the NSP NBI to manipulate service and endpoint parameters in the 5620 SAM mediation layer, even if the parameters are not defined in the abstract NSD models, and thus, deploy highly customized services.

Additional processing logic for custom attributes can be implemented by installing scripts in the 5620 SAM mediation layer. These system scripts can realize even complex provisioning operations, and they can be uploaded on the fly without restarting the 5620 SAM. The combination of custom attributes and scripts is therefore a very powerful method to achieve flexible and customized service provisioning in NSD. The 5620 SAM scripts can be further fine-tuned by professional service teams for any future needs.

Custom attributes are supported on service and endpoint objects for all IP services, including ELINE, ELAN, and L3VPN. Custom attributes can be set and modified by REST API users. Custom attributes can also be persistently configured in NSD custom attribute templates. Custom attributes cannot be configured from the NSD application.

Note: Values should not be supplied for custom attributes that are natively-configured by the NSP, such as the displayedName attribute. Providing values for custom attributes can cause some API requests to fail.

REST Patch

The REST Patch command allows for incremental updates to be applied to one or more of the basic attributes of an IP service (ELINE/ELAN/L3 VPN) and/or their corresponding endpoint(s). Any attribute that can be configured using an IP service/endpoint REST command can also be updated using the REST Patch command. For complex attributes (such as siteServiceQosProfile in L2EndpointRequest, routeTargets, routingBgp, routingStatic, and secondaryAddresses in L3EndpointRequest) the REST Patch command will replace the attribute's previously-provisioned value with the new value supplied by the user. There are no restrictions to the number of attributes that can be part of the REST Patch payload.

Filtering

When using the NSP REST API, filtering can be used to limit the number of objects returned by a GET request. Filters are defined in the query parameter of the URI within GET requests, and only objects with attributes that match the defined expression(s) are returned. The URI syntax supports expressions for many object attributes and also allows for logical combinations of more than one filter criteria. A filter is comprised of a series of attributes, identified by name, that are assigned values for which to filter.

Multiple filters can be separated by the '&' character (for AND). Values can be preceded by the '!' character for NOT. Multiple values are separated by commas. Parenthesis are required around values when using NOT.

The formal filtering notation is as follows:

```
filter      = expression {"&" expression}

expression    = attributeName "=" ["!("] value {"," value} [")"]
| "sortBy=" [-]attributeName

attributeName  = string

value       = string|number
```

The following notable information applies to filtering:

- Both attributeName= and attributeName=() specify that the value of attributeName is an empty string.
- attributeName=!() specifies that the value of attributeName is not an empty string.
- attributeName=(value1) and attributeName=value1 are equivalent.
- Only the first sortBy expression in a filter will take effect. Additional sortBy expressions will be ignored.
- Object attributes that have either a null value or a boolean value cannot be filtered out of a list, and therefore, will always appear in the results.
- Object attributes that are strings should not contain commas (,) as these are used as separators in an OR statement. String values are also case sensitive.
- A filtered search will ignore any unrecognized object attributes that are specified. A filtered search that contains a recognized numeric attribute, but an incomplete condition, will return an error. A filtered search that contains a recognized string attribute, but an incomplete condition, will return no results.

See the 1.3 “API collections” (p. 11) for more information about configuring a filter.

External applications notifications

The NSP's northbound interface allows its clients to receive notifications whenever the operational or administrative state of an NSP-managed service or endpoint changes. This simplifies synchronization with the NSP by removing the need for as periodic polling of the REST API. HTTPS Server Side Events (SSE) is used for the transport of these notifications, which are encoded in JSON according to the IETF RESTCONF protocol. See the following examples:

Notification for admin state change (service):

```
{
    "data": {
```

```
        "ietf-restconf:notification": {
            "eventTime": "2016-10-19T16:33:24.338Z",
            "nsp-service:service-state-change": {
                "admin-state": {
                    "old-value": "UP",
                    "new-value": "DOWN"
                },
                "uuid": "7095-86bc248d-f75a-438c-8de4-
d1221fe8711a"
            }
        }
    }
}
```

Notification for operational state change (endpoint):

```
{
    "data": {
        "ietf-restconf:notification": {
            "eventTime": "2016-10-19T16:33:24.425Z",
            "nsp-service:endpoint-state-change": {
                "operational-state": {
                    "old-value": "UP",
                    "new-value": "DOWN"
                },
                "uuid": "7107-4445bd7a-c741-4fdb-a9c5-
3141b85facb4"
            }
        }
    }
}
```

Notification for operational state change (service):

```
{
    "data": {
        "ietf-restconf:notification": {
            "eventTime": "2016-10-19T16:33:24.448Z",
            "nsp-service:service-state-change": {
                "operational-state": {
                    "old-value": "UP",
                    "new-value": "DOWN"
                },
                "uuid": "7095-86bc248d-f75a-438c-8de4-
d1221fe8711a"
            }
        }
```

```
    }
}
```

Note: A security token is required in order to subscribe to the notification stream. See the 1.3 “API collections” (p. 10) for more information about obtaining a token.

cURL, a generally-available tool for retrieving URLs, can be used to connect to the notification stream. The following command is used to subscribe (cURL uses HTTP GET method by default):

```
curl -k https://<server_address>:8543/ean/api/v3/notifications/
subscriber -H    'Authorization:<token>'
```

where

server address is the IP address of your NSP server

token is a previously-obtained authorization token

1.3 API collections

1.3.1 Postman collection

NSP REST API examples are available in a Postman collection that is provided alongside the full suite of NSP documentation. Postman itself can be downloaded from https://www.getpostman.com/apps. The collection of NSP REST API examples must be imported into Postman to be browsed. This collection has been organized so that folders contain individual workflows. The examples within the folders are presented in the order that they should be executed.

Example requests make use of Postman environments so that they can be executed against a running NSP instance. The following variables can be configured:

- *urlHost* - the hostname of the NSP server
- *token* - the token returned by the authorization API

In order for the example requests to be successfully executed, their UUIDs may need to be modified so as to reflect applicable values for the system that is being used. In most cases, the workflows have been organized so that UUIDs from earlier requests can be used in subsequent requests. Comments within each example also describe any prerequisites that may be necessary. For further details on using Postman, see the documentation at https://www.getpostman.com/docs.

1.3.2 HTML collection

NSP REST API examples are also available in an HTML collection that is provided alongside the full suite of NSP documentation. This collection has been organized so

that sections contain individual workflows. The examples within the sections are presented in the order that they should be executed. In order for the example requests to be successfully executed, their UUIDs may need to be modified so as to reflect applicable values for the system that is being used. In most cases, the workflows have been organized so that UUIDs from earlier requests can be used in subsequent requests. Comments within each example also describe any prerequisites that may be necessary.