NOKIA

FlowOne NEI for UIV ACTIONS

Release 20

Functional Description

Disclaimer

⚠ Important Notice on Product Safety

This product may present safety risks due to laser, electricity, heat, and other sources of danger.

Only trained and qualified personnel may install, operate, maintain or otherwise handle this product and only after having carefully read the safety information applicable to this product.

The safety information is provided in the Safety Information section in the “Legal, Safety and Environmental Information” part of this document or documentation set.

Nokia is continually striving to reduce the adverse environmental effects of its products and services. We would like to encourage you as our customers and users to join us in working towards a cleaner, safer environment. Please recycle product packaging and follow the recommendations for power use and proper disposal of our products and their components.

If you should have questions regarding our Environmental Policy or any of the environmental services we offer, please contact us at Nokia for any additional information.

1 About This Document

This document describes the NEI for FlowOne NEI for UIV ACTIONS.

1.1 Audience

This document is intended for the switch engineers, customer care staff and systems analysts who need to understand how this NEI works in the customer's provisioning system.

1.2 Terms and Concepts

The following sections list the abbreviations, terms and concepts used in the document.

Network Element Interface; NEI	A communication channel to a network element. A communication channel to a network element. NEI is an interface module in a Nokia product. Through a NEI, Nokia software can, for example, operate a network element, collect usage information from it or send control commands to it.

1.3 Related Documentation

The following documents give you additional information about this NEI:

- FlowOne NEI for UIV ACTIONS Release Notes
- FlowOne NEI for UIV ACTIONS Installation Guide

For more information on InstantLink, see InstantLink documentation.

2 System Overview

This chapter provides general information about the NEI and the environment in which the NEI is used.

2.1 Introduction to Unified Inventory

Unified Inventory (UIV) initiative is to build a single inventory solution that can provide discovery, reconciliation and inventory features for Fulfillment and Assurance of traditional and virtual networks.

2.2 Introduction to FlowOne NEI for UIV ACTIONS

The FlowOne NEI for UIV ACTIONS (hereinafter referred to as "UIV NEI") is designed to perform REST API query based on pre-configured template files. UIV NEI generates and sends REST request over HTTP/HTTPS connection to the target REST API server (for example, UIV server) and parse the REST response to generate response parameters based on template file. NEI template files should be pre-configured based on REST request and response format.

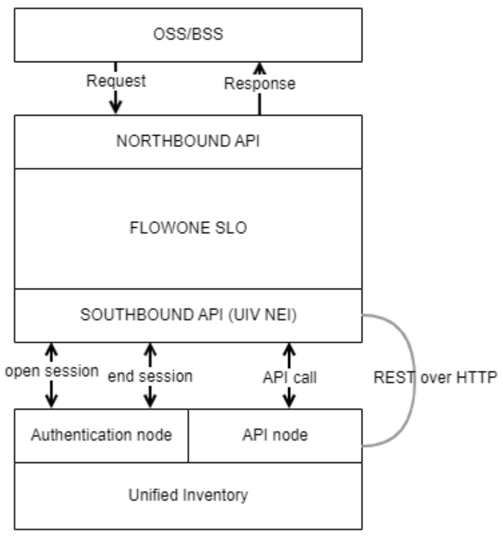

Figure 1. FlowOne NEI for UIV ACTIONS in the provisioning system

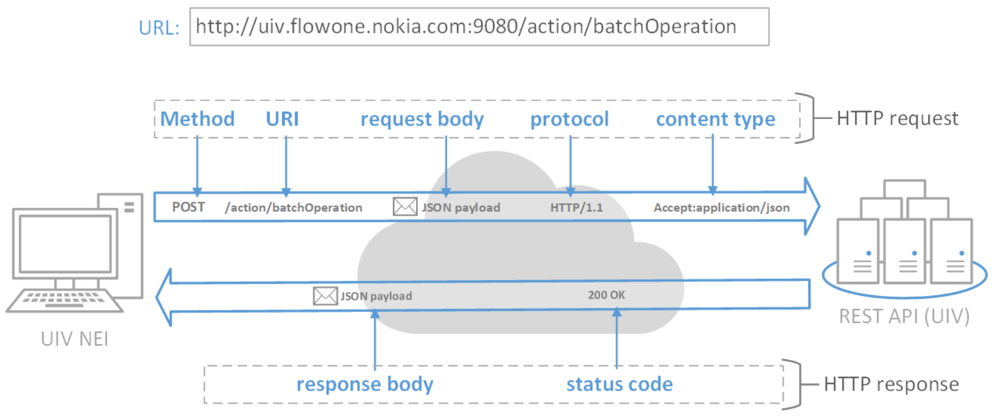

Figure 2. REST over HTTP communication

3 Login and Logout

UIV API is protected by OAuth2 authentication if `disable_verify_oauth2_access_token` is not set to `true` in RC configuration. All UIV API requests require OAuth2 token in the header. Two tokens will be provided upon successful login. The tokens are Access token and Refresh token.

Both Access token and Refresh token have configurable life span period at the authentication server side. NEI login operation will open a new session for new Refresh token and Access token. Within the same session, subsequent request towards UIV server will be attached with an Authorization header with a valid Access token. Expired Access token will be automatically renew by NEI.

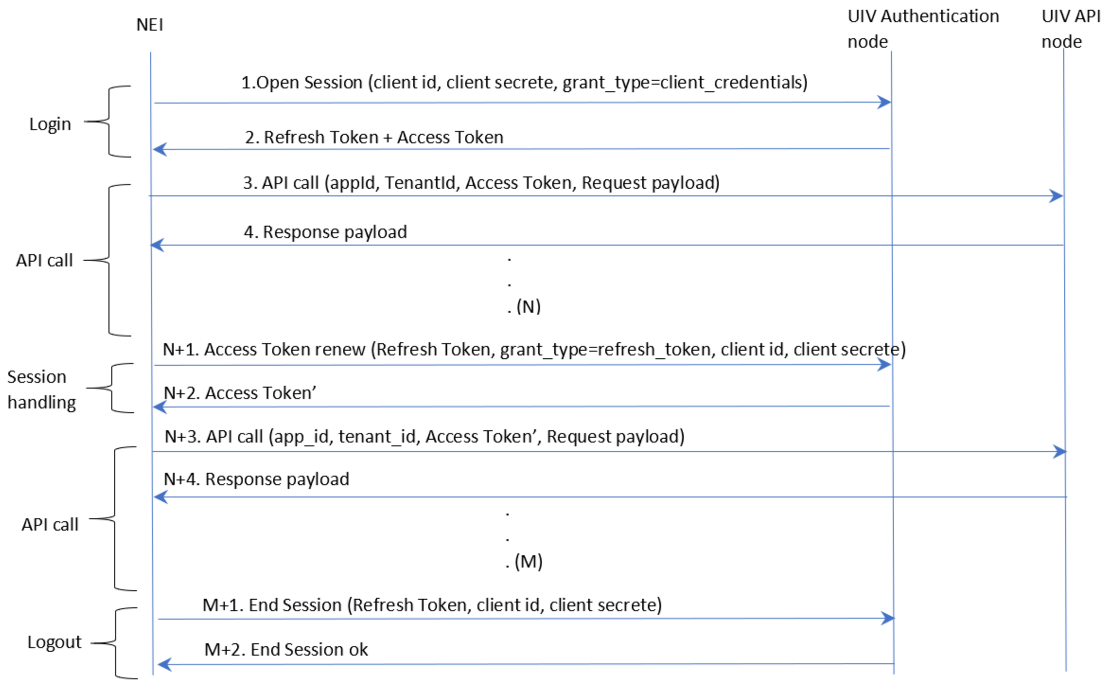
NEI
UIV Authentication node
UIV API node
Login
1.Open Session (client id, client secrete, grant_type=client_credentials)
2. Refresh Token + Access Token
API call
3. API call (appId, TenantId, Access Token, Request payload)
4. Response payload
. (N)
Session handling
N+1. Access Token renew (Refresh Token, grant_type=refresh_token, client id, client secrete)
N+2. Access Token'
API call
N+3. API call (app_id, tenant_id, Access Token', Request payload)
N+4. Response payload
. (M)
Logout
M+1. End Session (Refresh Token, client id, client secrete)
M+2. End Session ok

Figure 3. OAuth2 authentication

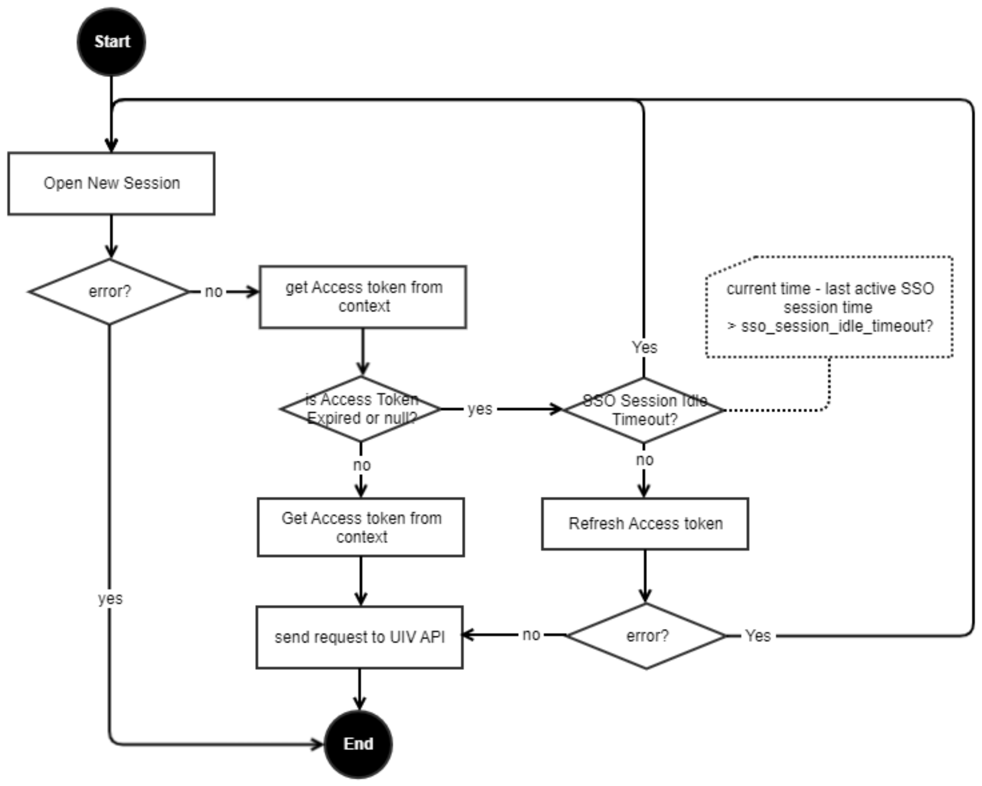
Start
Open New Session
error?
no
get Access token from context
is Access Token Expired or null?
yes
SSO Session Idle Timeout?
current time - last active SSO session time > sso_session_idle_timeout?
Yes
no
Get Access token from context
Refresh Access token
yes
send request to UIV API
no
error?
Yes
End

Figure 4. UIV NEI token handling

3.1 Login

Login is to open a session with authentication node to retrieve Refresh token and Access token which are required during the UIV API call.

Table 1. Task parameters

Task Parameter	Description	Values	M/O
`address1`	Token endpoint address in the format of `https://<host>:<port>/<token endpoint>`. **Note** This parameter is optional if `disable_verify_oauth2_access_token = true` in RC Configuration, else it is mandatory.	**Value type:** string **Example value:** `https://ext.flowone.nokia.local:28443/auth/realms/UIV/protocol/openid-connect/token`	O
`api_address1`	Address of the network element's API endpoint in the format of `https://<host>:<port>`.	**Value type:** string **Example value:** `https://ext.flowone.nokia.local:28443`	M

Task Parameter	Description	Values	M/O
`logout_address1`	End session endpoint address in the format of `https://<host>:<port>/<end session endpoint>`. **Note** This parameter is optional if `disable_verify_oauth2_access_token = true` in RC Configuration, else it is mandatory.	**Value type:** string **Example value:** `https://ext.flowone.nokia.local:28443/auth/realms/UIV/protocol/openid-connect/logout`.	O
`client_id1`	Oauth2.0 Client ID for client credential grant. **Note** This parameter is optional if `disable_verify_oauth2_access_token = true` in RC Configuration, else it is mandatory.	**Value type:** string	O
`client_secret1`	Oauth2.0 Client Secret for client credential grant. **Note** This parameter is optional if `disable_verify_oauth2_access_token = true` in RC Configuration, else it is mandatory.	**Value type:** string	O
`scope1`	Oauth2.0 Scope. **Note** This parameter is only applicable if `disable_verify_oauth2_access_token` is not `true` in RC Configuration. It is optional.	**Value type:** string	O
`app_id1`	The appId of the request message's header towards UIV API endpoint. **Note** This parameter is optional if `disable_verify_oauth2_access_token = true` in RC Configuration, else it is mandatory.	**Value type:** string	O
`tenant_id1`	The tenantId of the request message's header towards UIV API endpoint. **Note** This parameter is optional if `disable_verify_oauth2_access_token = true` in RC Configuration, else it is mandatory.	**Value type:** string	O

All the above parameters are to be configured in the Instantlink Network Model. For more information, see *FlowOne NEI for UIV ACTIONS Installation Guide.*

The following shows example login communication with the UIV’s Authentication server.

Login header example:

```
POST /uiv/xpon/action/batchOperation HTTP/1.1
host: "ext.flowone.nokia.local:28443"
content-type:"application/x-www-form-urlencoded"
authorization:"Basic
UVlLWUFQUDo3ZGYxOGMwZC1kNGM1LTQ31jEtYjk1OS2hZkj3MjY5NGRYhjA="
```

Response header:

```
content-type:"application/json"
content-length:"2562"
cache-control:"no-store"
set-cookie:"KC_RESTART=; Version=1; Expires=Thu, 01-Jan-1970
00:00:10 GMT; Max-Age=0; Path=/auth/realms/UIV/; HttpOnly;
Secure"
pragma:"no-cache"
date:"Fri, 25 Oct 2019 01:24:11 GMT"
x-kong-upstream-latency:"84"
x-kong-proxy-latency:"174"
via:"kong/0.13.1"
```

Response body:

```
access_token:"<SOME VALUE>"
expires_in:60
refresh_expires_in:0
refresh_token:"<SOME VALUE>"
token_type:"bearer"
not-before-policy:0
session_state:"13ff2e66-cade-4eae-9702-352298ed5f12"
scope:"profile offline_access email"
```

The response contains Access token and Refresh token.

3.2 Refresh Access Token

The following shows the example of UIV NEI communication with the UIV's Authentication server on how to refresh the access token.

Request header:

```
POST /auth/realms/UIV/protocol/openid-connect/token HTTP/1.1
Host: ext.flowone.nokia.local:28443
content-type: application/x-www-form-urlencoded
```

Request body:

```
"grant_type=refresh_token&refresh_token=<SOME
VALUE>&client_id=UIV-NEI&client_secret=1ff17c3d-d5a0-b7c1-
b959-cd182674dab0"
```

Response header:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 2496
Cache-Control: no-store
Pragma: no-cache
Date: Fri, 08 Nov 2019 06:57:56 GMT
X-Kong-Upstream-Latency: 10
X-Kong-Proxy-Latency: 0
Via: kong/0.13.1
```

Response body:

```
{
"access_token":"<SOME VALUE>",
"expires_in":60,
"refresh_expires_in":120,
"refresh_token":"<SOME VALUE>",
"token_type":"bearer",
"not-before-policy":1573177721,
"session_state":"fea322e8-16de-49e1-a40a-f8e1113ebb5c",
"scope":"profile email"
}
```

The response contains Access token and Refresh token.

3.3 UIV API Call

The following shows the example of UIV NEI communication with the UIV's API node.

Request header:

```
POST  /uiv/xpon/action/createDeviceInterface HTTP/1.1
content-type:"application/json"
tenantId: UIV
appId: UIV-NEI
Authorization: Bearer <ACCESS TOKEN VALUE>
```

Request body:

```
{
     "interfaceType": "NNI_HSI",
     "context": "A",
     "endUserLocationName": "CDFF_CUST_15102019_01",
     "cvlan": "1001",
     "svlan": "2101"
}
```

Response Header:

```
{Transfer-Encoding=[chunked], X-Kong-Upstream-Latency=[333],
X-Kong-Proxy-Latency=[170], Date=[Thu, 31 Oct 2019 02:14:51
GMT], Via=[kong/0.13.1], Content-
Type=[application/json;charset=utf-8]}
```

Response Body:

```
{

     "logicalInterfaceName":
"CDFF_CUST_15102019_01_NNI_HSI_LogicalInterface",

     "logicalInterfaceId": "138cd17a-3150-4f63-b82a-
fe1dcffcff14"

}
```

3.4 Logout

Logout is to close the session opened during login.

The following shows the example of logout communication with the UIV's Authentication server.

Logout header example:

```
POST /auth/realms/UIV/protocol/openid-connect/logout HTTP/1.1
content-type:"application/x-www-form-urlencoded"
```

Logout body:

```
"refresh_token=<SOME VALUE>&client_secret=4df17c0d-d5c7-17b1-
b959-af972684dab0&client_id=UIV-NEI"
```

Response header:

```
{X-Kong-Upstream-Latency=[35], X-Kong-Proxy-Latency=[2],
Date=[Thu, 31 Oct 2019 02:15:10 GMT], Via=[kong/0.13.1],
Content-Type=[text/plain; charset=UTF-8]}
```

4 Provisioning Task

This chapter describes the tasks supported by the UIV. The task implemented for UIV is Modify.

4.1 Modify

Use the Modify task type to perform an operation.

Figure 5 illustrates the process:

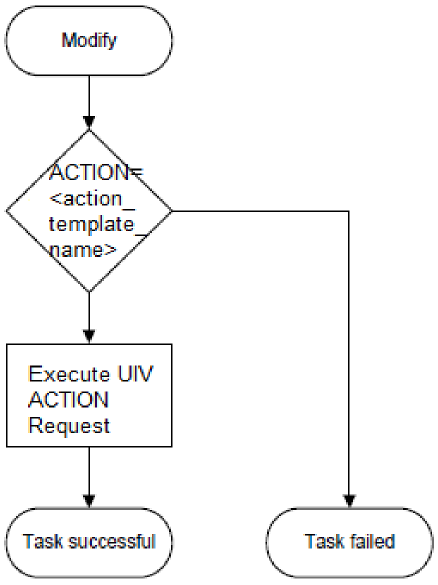

Figure 5. Modify procedure

4.1.1 runAction

runAction is the only declared method of Connection class. This function is used to execute an action in UIV.

5 Template Overview

UIV NEI is a template based NEI. It consists of two type of templates, which are the action template and object template. This NEI is responsible to generate request to be sent to UIV and process response received from the interface according to the action templates provided.

5.1 Action Template and Object Template

This section describes action template and object template format details. Action template defines the details required to perform an UIV action or CRUD API call, while object template defines the objects schema referred by action templates.

Action template is a text file with the following skeleton:

- HTTP header
- request template
- response template
- response error code mapping

The HTTP header allows user to define different properties of HTTP header for HTTP communication, namely method, URI, and content type.

The request body are defined as JSON format within the request template section. The value for each element in the request body can be a fixed or a variable. Variable value definition can be achieved with some customized syntax, while the corresponding value will be mapped from the NEI input parameters. For more information Refer section 5.1.2 request template for details.

The response body received by NEI can be parsed into NEI response parameters, JSONPath syntax is used for the parameter mapping within the response template section of the action template.

In cases of request failure, the HTTP error code can be mapped into customized error code in the response error code mapping section of the action template.

Note For the sample of action template and object template, see *Appendix A: Action Template* and *Appendix B: Object Template* which are delivered as part of this NEI package.

Following annotation will be used to represent different section of an Action template.

Table 2. Action Template Annotation

Annotation	Description	Value	M/O
`@HTTP_METHOD`	HTTP method for the REST request.	**Value type:** String **Example values:** • `POST`	M
`@HTTP_URI`	HTTP URI for REST API resource.	**Value type:** String **Example value:** `$TEMPLATE_HTTP_URI$` **Note** This is a mandatory variable and is required to be inserted as the NEI request parameter. The annotation `@HTTP_URI` supports fixed or variable parameter excluding object template reference. For more information, see Section 5.1.2.1 *Mandatory Variable* and Section 5.1.2.2 *Optional Variable*.	M
`@HTTP_CONTENT_TYPE`	HTTP content type of the resource.	**Value type:** String **Example value:** `application/json`	M
`@REQUEST_TEMPLATE`	Request template for Request body definition.	For more information, see Section 5.1.2 *Action Template*.	M
`@RESPONSE_TEMPLATE`	Response template for Response parameter mapping.	Empty value is allowed. For more information, see Section 5.1.3 *Response Template*.	M
`@ERROR_CODE_MAPPING`	Response error code mapping for HTTP error code mapping to NEI error code.	Empty value is allowed. For more information, see Section 5.1.4 *Response Error Code Mapping*.	M

Below is the example of a basic action template which contains fixed value, variable, and optional object:

Action file name: `MultiCRUD.action`

```
@HTTP_METHOD: "POST"
@HTTP_URI: "/action/batchOperation"
@HTTP_CONTENT_TYPE: "application/json"

@REQUEST_TEMPLATE:
{
```

```
          "cargos": [{
             "kind" : "$KIND$" ,
             "type" : "$TYPE$" ,
             "objects" : "$Template(SERVICE, min=0, max=*)"
          }]
}

@RESPONSE_TEMPLATE:
{
          ID=$[*].cargo.objects[*].id
          DESCRIPTION=$[*].cargo.objects[*].description
}

@ERROR_CODE_MAPPING:
{
          500, ERR500, Internal Server Error
}
```

Object file name: `SERVICE.tmpl`

```
{
          "id": "$ID?",
          "state": "$STATE?",
          "context": "$CONTEXT?",
          "localName": "$LOCALNAME?",
          "globalName": "$GLOBALNAME?",
          "description": "$DESCRIPTION?",
          "contained" : "$Template(SERVICE, min=0, max=*)",
          "properties" : "$Template(PROPERTIES, min=0, max=*,
singleobj=true)"
}
```

Object file name: `PROPERTIES.tmpl`

```
{
          "$NAME$": "$VALUE$"
}
```

Below is an example of the request task parameter for the above templates:

```
Run task 100_1
task_type=modify
ACTION=TEST
REQ_TYPE=2
KIND=Create
TYPE=com.nokia.nsw.uiv.model.common.party.Customer
DESCRIPTION=This is testing
LOCALNAME=001
SERVICE[1].CONTEXT=001
SERVICE[1].LOCALNAME=HSI
SERVICE[1].DESCRIPTION=This is highspeedinternet
SERVICE[1].PROPERTIES[1].NAME=Catalog Item Version
```

```
SERVICE[1].PROPERTIES[1].VALUE=1.0
SERVICE[1].PROPERTIES[2].NAME=Transaction Type
SERVICE[1].PROPERTIES[2].VALUE=Local
End
```

Below is an example of a response return to UIV NEI:

```
[{
     "cargo": {
          "identifier": null,
          "variables": null,
          "condition": null,
          "kind": "Create",
          "objects": [{
               "note": [],
               "contactMedium": null,
               "usedResource": [],
               "displayName": "001,HSI",
               "serviceConsumer": [],
               "description": "This is highspeedinternet",
               "updatedDate": null,
               "validFrom": null,
               "subscription": [],
               "mode": null,
               "reference": [],
               "id": "72c8ae64-9bad-45ae-8a82-e5d481fcbb0f",
               "place": [],
               "updatedBy": null,
               "kind": null,
               "entityType": [],
               "_type": "customer",
               "contextId": null,
               "isRootNetworkElement": null,
               "globalName": "001,HSI",
               "ownedResource": [],
               "createdDate": 1565080994231,
               "createdBy": null,
               "service": [],
               "localName": "HSI",
               "validUntil": null,
               "properties": {
                    "Catalog Item Version": "1.0",
                    "Transaction Type": "Local"
               },
               "party": null
          }],
          "name": null,
          "type":
"com.nokia.nsw.uiv.model.common.party.Customer"
     },
     "message": null,
     "status": "Success"
}]
```

5.1.1 HTTP Header

The UIV NEI rely on the following annotation to send REST request:

- `HTTP_METHOD`
- `HTTP_URI`
- `HTTP_CONTENT_TYPE`

For more information, see *Table 2. Action Template Annotation.*

5.1.2 Action Template

UIV NEI based on action template to construct REST request. Action template defines the request body of the REST request. Only Single REST operation or request can be sent within a single UIV NEI call. In cases where multiple operations are required to be sent together, UIV's `batchOperation` functionality shall be considered.

Nested template call is supported by UIV NEI where Action template can call an object template and object template can make reference to another object template. There is no depth limit for the nested template call functionality.

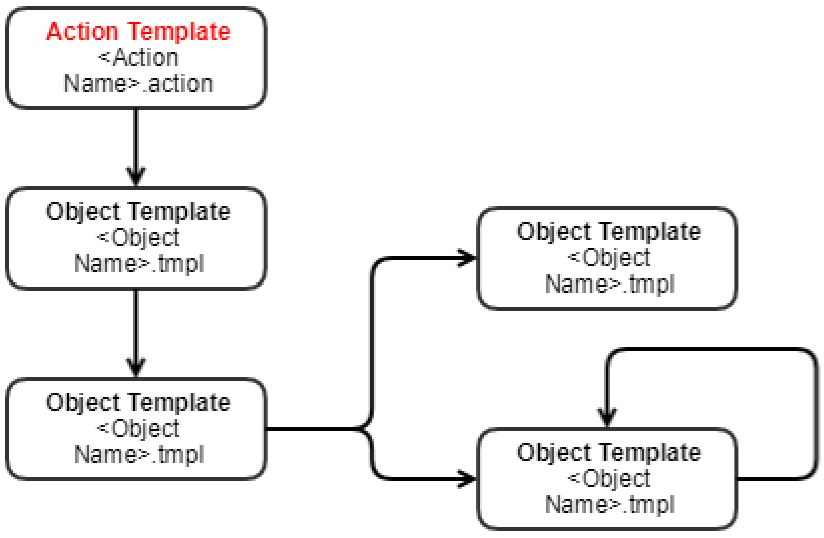

Figure 6. Template recursive call

UIV NEI requires the action template and object template to be stored in the following location and it is configurable in RC configuration:

```
<installdir>/sas/bin/macro_server/java/UIV/ACTIONS/ANY/templat
e
```

The following naming convention is recommended to keep UIV NEI organised and to ensure smooth integration between multiple technology templates.

Table 3. Template file naming

Template Type	Naming convention	Description	Example

Action template	`<Domain>_<Action Name>.action`	`<Domain>`
		Type, group, project or category name which allows user to identify a series of action files and prevent overlapping with another template deployment.
		
		`<Action name>`
		Unique name to identify an action template and is recommended to use the functionality name such as `CreateService` and `CreateEndUserLocation`.
		Action template cannot be nested and called by another action template, thus action template name should always be unique.
			`XPON_ConnectEndUserLocation.action`
			`MPLS_CreateStaticRoute.action`
			`Mano_GetVNFPlacementGroupParameters.action`
			`TelcoA_GetServicePath.action`
			`TelcoA_GetPort.action`

Object template	`<Domain>_<Object Name>.tmpl`	`<Domain>`
		Type, group, project or category name which allows user to identify a series of action files and prevent overlapping with another template deployment. This prefix could be omitted when a common object template creation is intended.
		
		`<Object Name>`
		Unique name to identify an action template and is recommended to use the UIV Object name. Object template can be used by another action template. Thus, action template name should be based on UIV Object schema name which is unique in the UIV.
			`Service.tmpl`
			`ServiceCargo.tmpl`
			`ServiceObject.tmpl`
			`Properties.tmpl`
			`TelcoA_Service.tmpl`
			`TelcoA_ServiceCargo.tmpl`
			`TelcoA_ServiceObject.tmpl`

The action template driven feature generates the REST request body by referring to the action template file that is specified in the request.

The flow of action template driven feature:

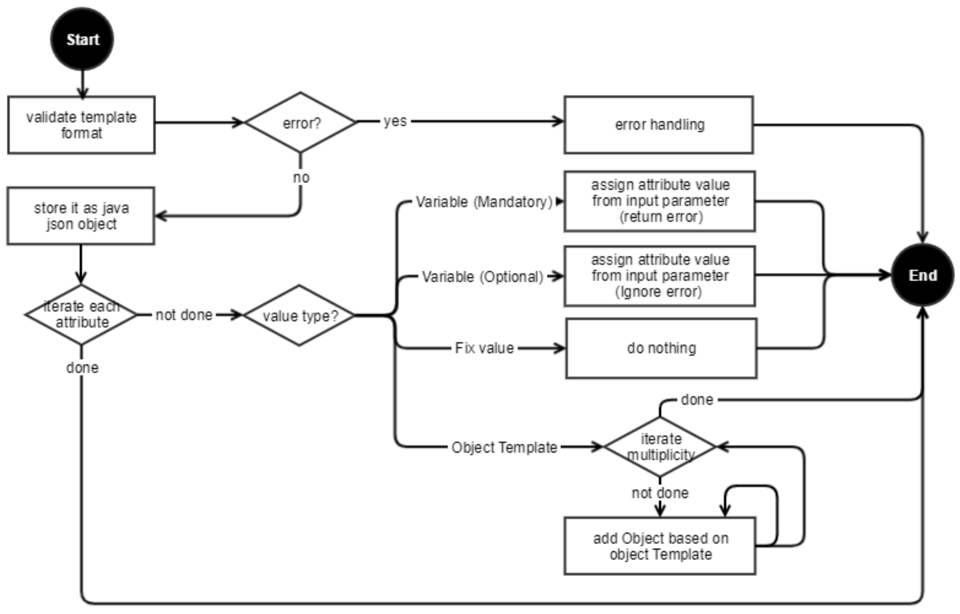

Figure 7. Request template processing

1. The NEI loads all the task parameters given in the request into an argument list. The NEI grabs the specified action template file name from the task parameter in the request.

2. The NEI loads the action template file specified in the request from template directory.

3. The NEI will read the action template and replace all the required values with the value stored in the argument list. If the request from the action template is a nested template, this process will be repeated until it loads all the request templates. The content of the nested request template will be kept in a temporary memory.

4. When all nested request template files are loaded, the NEI combines the content of nested template with request template to generate the proper request body JSON.

5. The NEI uses Spring Web 4.3.11 API to wrap the request body JSON. After wrapping the request body JSON, the request will be sent.

5.1.2.1 Mandatory Variable

Mandatory variable is defined with the enclosed dollar sign (`$`) within a variable name. For example, `$KIND$`. The corresponds request task parameter is `KIND=Create`.

5.1.2.2 Optional Variable

Optional variable is defined with the enclosed dollar sign (`$`) and question mark (`?`) within a variable name. For example, `$DESCRIPTION?`. The corresponds request task parameter is `DESCRIPTION= This is testing`.

5.1.2.3 Object Template Reference

To include an object template into the main action template by providing the template name as a variable. The variable name used for a sub template must have a suffix `.tmpl` (configurable from RC configuration) and it is case-sensitive.

Syntax:

```
$Template(<Template name>, <min occurance>, <max occurance>,
<Single object>)
```

Example:

```
"$Template(SERVICE, min=0, max=*)"
```

The `$Template` supports four parameters:

- Template name
  - Name of the sub template and it is case-sensitive.
- Minimum occurrence
  - Minimum number of times `SERVICE` to be constructed.
  - If value is set to `0`, it is represented as an optional key.
  - If value is set to `1`, it is represented as a mandatory. NEI performs checking on request task parameter. For example, at least one `SERVICE[1]` is found from request task parameter.
- Maximum occurrence
  - Maximum number of times `SERVICE` to be constructed.
  - If value is set to `*`, it is represented as a list of objects to be created without cap.
  - If value is set to `1`, it is represented as a non-list of objects to be created.
  - If value is set to more than 1, it is represented as total cap number of objects in list to be created.
- Single object indicator (optional)
  - Valid value: `true` or `false`.
    - If value is set to `true`, content inside the sub template will repeat `X` times within a single object based on the provided request task

parameter. Based on the example above, template `PROPERTIES` repeated two times.

5.1.2.4 Mixed Hardcoded Value with Optional and Mandatory Variable

The NEI supports parameter value with string appended into it. To use this feature, the optional or mandatory variable must be encased with curly braces, `{` and `}`. An example of this parameter value is `/uiv/xpon/path/{$OPTIONAL?}/` where the corresponding value will be:

for `OPTIONAL = ""` , the `value = /uiv/xpon/path//`

for `OPTIONAL = "gpon"`, the `value = /uiv/xpon/path/gpon/`

Note Using both the mixed string feature with curly braces is not supported. For example, `{$OPTIONAL}` and `$OPTIONAL$` (non-mixed string as described in Section 5.1.2.1 and 5.1.2.2).

5.1.3 Response Template

Every request template can have a corresponding response template. However, the format for the response template is different compare to request template. The content of the response template can be left empty if there is no response parameter required to be returned from NEI.

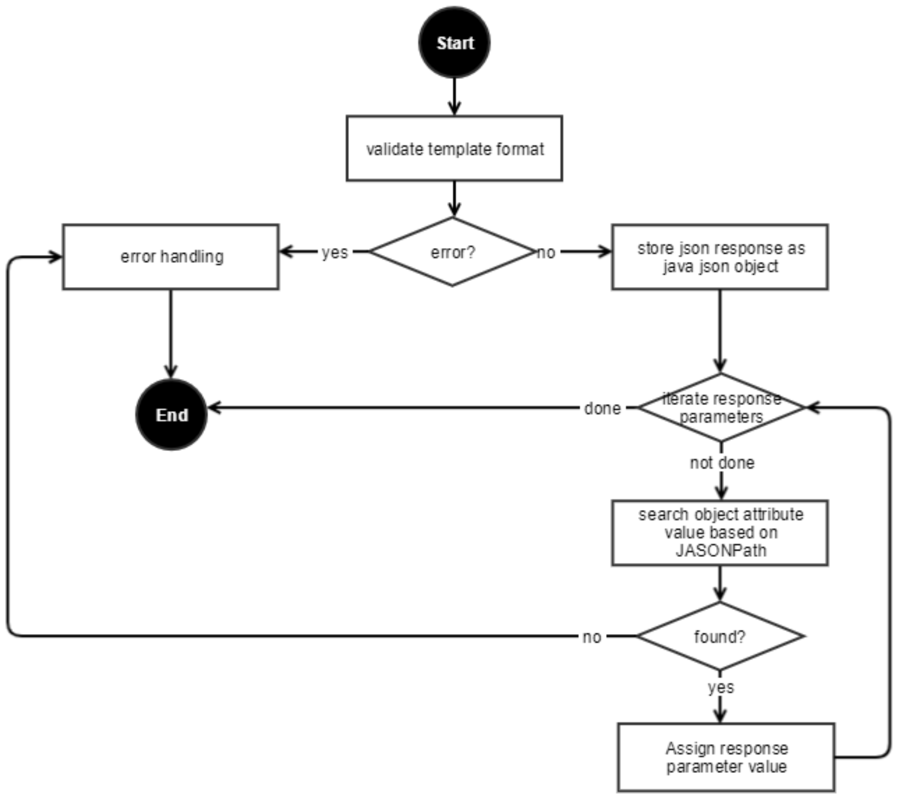
Start
validate template format
error?
yes
no
error handling
store json response as java json object
End
done
iterate response parameters
not done
search object attribute value based on JASONPath
no
found?
yes
Assign response parameter value

Figure 8. Response template parser logic

The content for UIV NEI response template basically consists of two parts, which are the variable name to be used for that element (left side) and the JSONPath expression (right side). The following format must be used to configure the response:

```
<RESPONSE_PARAM_NAME> = <JSONPATH_EXPRESSION>
```

If the response parameter name contains `_<n>` as a suffix, for example `SUB_NAME_<n>`, the response output will always append numerical value starting from 1. For example, `SUB_NAME_1 = Subscriber_001`.

Field	Description	Values	M/O
`RESPONSE_PARAM_NAME`	The response parameter. Use `<n>` to denote multiple values returned.	**Value type:** String **Example values:** • `SUB_ID` • `SUB_ID_<n>`	M
=	Mapping separator. Left side is the recipient of the attribute value and right side is the attribute name syntax guiding the parser logic to search for certain attribute in the json response object.	=	M
`JSONPATH_EXPRESSION`	Defines the JSONPath expression to parse through the response message body.	**Value type:** String `$[*].id`	M

Note If `RESPONSE_PARAM_NAME` is without `<n>` and the returned response contains more than 1 result, NEI will output `RESPONSE_PARAM_NAME` with `<n>`.

Whenever a `RESPONSE_PARAM_NAME` is provided, the NEI will grab the value of the `RESPONSE_PARAM_NAME` and evaluate from JSONPath API to retrieve the element's value. If the element is found, the value will be stored in `RESPONSE_PARAM_NAME`. The NEI sends the value with the `RESPONSE_PARAM_NAME` back to OSS/BSS.

UIV NEI uses JSONPath expressions from JSONPath API to parse the JSON response from UIV. JSONPath expressions always refer to a JSON structure in the same way as XPath expression are used in combination with an XML document. The root object in JSONPath is always referred to as `$` regardless if it is an object or array.

JSONPath expressions can use the dot–notation

```
$[*].id
```

or the bracket–notation

```
$[*]['id']
```

UIV NEI is able to parse a list of string from the returned value of JSONPath expressions.

Example result of the response template in action template file:

```
ID 001
DESCRIPTION This is testing
```

Note For more information on how to use JSON expression, see Appendix C: *JSONPath Expression*.

Note For more information on JSONPath library, please visit https://github.com/json-path/JsonPath.

5.1.4 Response Error Code Mapping

In the event of request processing failure from UIV, REST response message will contain HTTP status code and message. This code and message can be mapped to NEI error code and error message which will be returned as NEI response parameter, for example, `MESSAGE_ID` and `MESSAGE`.

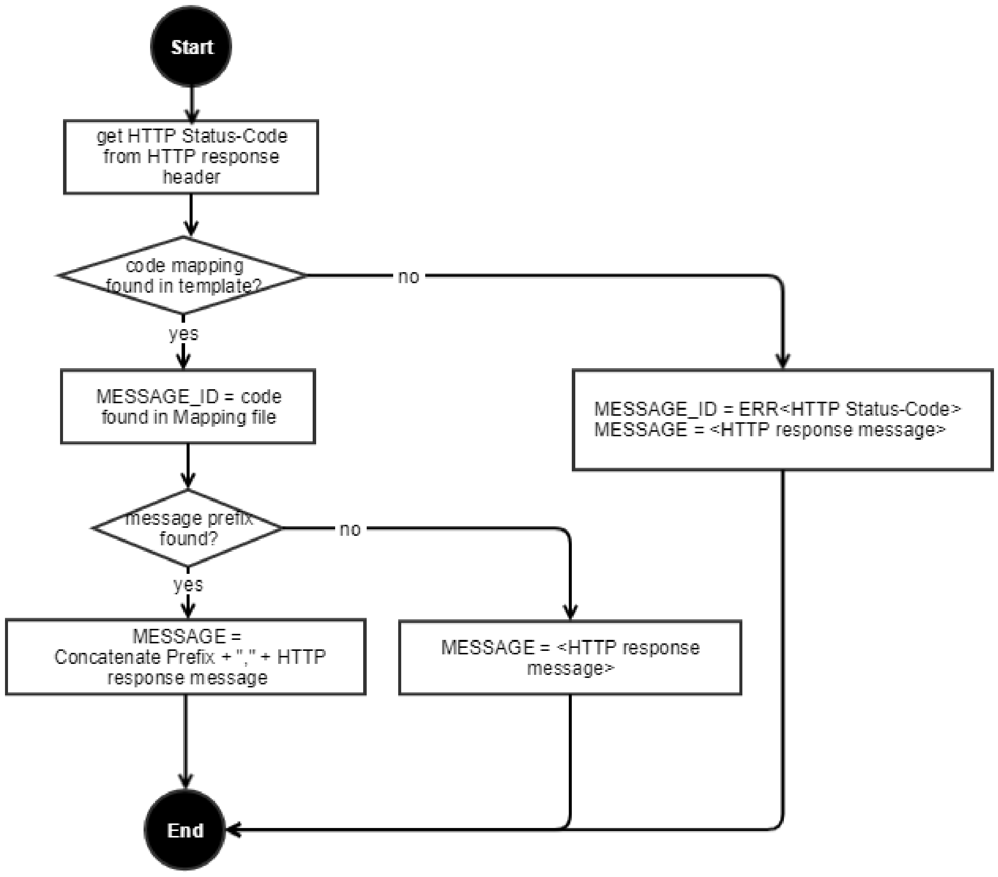
Start
get HTTP Status-Code from HTTP response header
code mapping found in template?
no
yes
MESSAGE_ID = code found in Mapping file
MESSAGE_ID = ERR<HTTP Status-Code>
MESSAGE = <HTTP response message>
message prefix found?
no
yes
MESSAGE = Concatenate Prefix + "," + HTTP response message
MESSAGE = <HTTP response message>
End

Figure 9. Response Code Mapping Logic

```
<HTTP Status Code>,<Customize Error Code>,<Customize Error
Description>
```

If HTTP status code is not found in error code mapping section, a default error code will follow the HTTP status code with a defined prefix `ERR` and displayed in `MESSAGE_ID`. The `MESSAGE` will only display error description from HTTP.

Lookup Parameter	Description	Values	M/O
`HTTP Status Code`	HTTP status code from response.	**Value type:** String **Example value:** `500`	M
`Customize Error Code`	Defines customize error code to be returned.	**Value type:** String **Example value:** `ERR500`	M
`Customize Error Description`	Defines a prefix description of the error code.	**Value type:** String **Example value:** `Internal Server Error`	O

Example of response code mapping:

Response Code mapping	Error Response from UIV	NEI Response

`@ERROR_CODE_MAPPING:`
`{`
`404,BST0001,Resource`
`}`
	`{`
	`    "timestamp": 1555038477158,`
	`    "status": 404,`
	`    "error": "Not Found",`
	`    "message": "Not Found",`
	`    "path": "/api/servicesq"`
	`}`
		`< 7200 NEW MESSAGE_ID BST0001`
		`< 7200 NEW MESSAGE Resource, Not Found`

Appendix A: Action Template

File name: `MultiCRUD.action`

```
@HTTP_METHOD: "POST"
@HTTP_URI: "$TEMPLATE_HTTP_URI$"
@HTTP_CONTENT_TYPE: "application/json"

@REQUEST_TEMPLATE:
{
         "cargos": "$Template(CARGO, min=1, max=*)"
}

@RESPONSE_TEMPLATE:
{
  ID=$[*].cargo.objects[*].id
  LOCALNAME=$[*].cargo.objects[*].localName
  GLOBALNAME=$[*].cargo.objects[*].globalName
  DESCRIPTION=$[*].cargo.objects[*].description
  STATE=$[*].cargo.objects[*].state
}

@ERROR_CODE_MAPPING:
{
}
```

Appendix B: Object Template

File name: `ServiceCargo.tmpl`

```
{
         "kind" : "$KIND$" ,
         "type" : "$TYPE$" ,
         "objects" : "$Template(ServiceObject, min=1, max=*)"
}
```

File name: `ServiceObject.tmpl`

```
{
         "state": "$STATE?",
         "description": "$DESCRIPTION?",
         "context": "$CONTEXT?",
         "localName": "$LOCALNAME?",
         "_identifier": "$IDENTIFIER?",
         "service" : "$Template(Service, min=0, max=*)",
         "properties" : "$Template(Properties, min=0, max=*,
         singleobj=true)"

}
```

File name: `Service.tmpl`

```
{
         "state": "$STATE?",
         "context": "$CONTEXT?",
         "localName": "$LOCALNAME?",
         "_identifier": "$IDENTIFIER?",
         "description": "$DESCRIPTION?",
         "contained" : "$Template(Service, min=0, max=*)",
         "properties" : "$Template(Properties, min=0, max=*,
singleobj=true)"
}
```

File name: `Properties.tmpl`

```
{
         "$NAME$": "$VALUE$"
}
```

Appendix C: JSONPath Expression

JSONPath Expression

JSONPath expressions always refer to a JSON structure in the same way as XPath expression are used in combination with an XML document. The root object in JSONPath is always referred to as $ regardless if it is an object or array.

JSONPath expressions can use the dot–notation

```
$.store.book[0].title
```

or the bracket–notation

```
$['store']['book'][0]['title']
```

Operators

Operator	Description
`$`	The root element to query. This starts all path expressions.
`@`	The current node being processed by a filter predicate.
`*`	Wildcard. Available anywhere a name or numeric are required.
`..`	Deep scan. Available anywhere a name is required.
`.<name>`	Dot-notated child.
`['<name>' (, '<name>')]`	Bracket-notated child or children.
`[<number> (, <number>)]`	Array index or indexes.
`[start:end]`	Array slice operator.
`[?(<expression>)]`	Filter expression. Expression must evaluate to a Boolean value.

Functions

Functions can be invoked at the tail end of a path. The input to a function is the output of the path expression. The function output is dictated by the function itself.

Function	Description	Output
`min()`	Provides the min value of an array of numbers.	Double
`max()`	Provides the max value of an array of numbers.	Double
`avg()`	Provides the average value of an array of numbers.	Double
`stddev()`	Provides the standard deviation value of an array of numbers.	Double
`length()`	Provides the length of an array.	Integer

Filter Operators

Filters are logical expressions used to filter arrays. A typical filter would be `[?(@.age > 18)]` where `@` represents the current item being processed. More complex filters can be created with logical operators `&&` and `||`. String literals must be enclosed by single or double quotes `[?(@.color == 'blue')]` or `[?(@.color == "blue")]`.

Operator	Description
`==`	Left is equal to right (note that 1 is not equal to '1').
`!=`	Left is not equal to right.
`<`	Left is less than right.
`<=`	Left is less or equal to right.
`>`	Left is greater than right.
`>=`	Left is greater than or equal to right.
`=~`	Left matches regular expression `[?(@.name =~ /foo.*?/i)]`.
`in`	Left exists in right `[?(@.size in ['S', 'M'])]`.
`nin`	Left does not exists in right.
`subsetof`	Left is a subset of right `[?(@.sizes subsetof ['S', 'M', 'L'])]`.
`anyof`	Left has an intersection with right `[?(@.sizes anyof ['M', 'L'])]`.
`noneof`	Left has no intersection with right `[?(@.sizes noneof ['M', 'L'])]`.
`size`	Size of left (array or string) should match right.
`empty`	Left (array or string) should be empty.

Path Examples:

Given the JSON

```
{
    "store": {
        "book": [
            {
                "category": "reference",
                "author": "Nigel Rees",
                "title": "Sayings of the Century",
                "price": 8.95
            },
            {
                "category": "fiction",
                "author": "Evelyn Waugh",
                "title": "Sword of Honour",
                "price": 12.99
            },
            {
                "category": "fiction",
                "author": "Herman Melville",
                "title": "Moby Dick",
                "isbn": "0-553-21311-3",
                "price": 8.99
            },
            {
                "category": "fiction",
                "author": "J. R. R. Tolkien",
                "title": "The Lord of the Rings",
                "isbn": "0-395-19395-8",
                "price": 22.99
            }
        ],
        "bicycle": {
            "color": "red",
            "price": 19.95
        }
    },
    "expensive": 10
}
```

JSONPath	Result
`$.store.book[*].author`	The authors of all books.
`$..author`	All authors.
`$.store.*`	All data from both books and bicycles.
`$.store..price`	Return all the attribute data (price).
`$..book[2]`	The third data (book).
`$..book[-2]`	The second to last data (book).
`$..book[0,1]`	The first two data (book).
`$..book[:2]`	All books from index 0 (inclusive) until index 2 (exclusive).
`$..book[1:2]`	All books from index 1 (inclusive) until index 2 (exclusive).
`$..book[-2:]`	Last two books.
`$..book[2:]`	Book number two from tail.
`$..book[?(@.isbn)]`	All books with an ISBN number.
`$.store.book[?(@.price < 10)]`	All books in store cheaper than 10.
`$..book[?(@.price <= $['expensive'])]`	All books in store that are not `expensive`.
`$..book[?(@.author =~ /.*REES/i)]`	All books matching regex (ignore case).
`$..*`	Return all the data in the JSON.
`$..book.length()`	The number of data (books).