ONAP External API – Northbound API

Table of Contents

[Modification table 3](#_Toc508185886)

[Assumptions 4](#_Toc508185887)

[Service Catalog: 4](#_Toc508185888)

[Service order 4](#_Toc508185889)

[Questions 5](#_Toc508185890)

[Use-Case 7](#_Toc508185891)

[vFirewall 7](#_Toc508185892)

[Global Architecture 8](#_Toc508185893)

[External API – internal attributes 8](#_Toc508185894)

[TMF API 10](#_Toc508185895)

[TMF ServiceOrder API 10](#_Toc508185896)

[TMF Resource Model: 10](#_Toc508185897)

[Swagger 11](#_Toc508185898)

[ServiceInventory API 12](#_Toc508185899)

[Resource Model: 12](#_Toc508185900)

[Swagger 13](#_Toc508185901)

[ServiceCatalog API 14](#_Toc508185902)

[Resource Model: 14](#_Toc508185903)

[Swagger 14](#_Toc508185904)

[Service Catalog Mapping 15](#_Toc508185905)

[Sources 15](#_Toc508185906)

[API Description 15](#_Toc508185907)

[Swagger: 15](#_Toc508185908)

[Lifecycle: 15](#_Toc508185909)

[API Operation provided: 15](#_Toc508185910)

[ServiceSpecification retrieval limitation: 15](#_Toc508185911)

[Postman Collection 15](#_Toc508185912)

[Mapping rules 15](#_Toc508185913)

[Service Order 20](#_Toc508185914)

[Sources 20](#_Toc508185915)

[API Description 20](#_Toc508185916)

[Swagger: 20](#_Toc508185917)

[Lifecycle: 20](#_Toc508185918)

[API Operation provided: 21](#_Toc508185919)

[Postman Collection: 22](#_Toc508185920)

[POST Service Order High Level view 22](#_Toc508185921)

[Detailed view step by step 23](#_Toc508185922)

[Retrieve ServiceOrder 32](#_Toc508185923)

[Service Inventory Mapping 34](#_Toc508185924)

[Sources 34](#_Toc508185925)

[API Description 34](#_Toc508185926)

[Swagger 34](#_Toc508185927)

[Lifecycle: 34](#_Toc508185928)

[API Operation provided: 34](#_Toc508185929)

[GET Service Inventory (list) 34](#_Toc508185930)

[GET Product Inventory by id 35](#_Toc508185931)

[ONAP API Description 38](#_Toc508185932)

[SDC API 38](#_Toc508185933)

[API resource model: 38](#_Toc508185934)

[SO API 39](#_Toc508185935)

[API resource model: 39](#_Toc508185936)

[A&AI API 40](#_Toc508185937)

[API resource model: 40](#_Toc508185938)

# Modification table

|  |  |
| --- | --- |
| Date | Modification |
| xxxxx | Initial Version |
|  |  |

# Assumptions

Design is based on ONAP API described on ONAP readTheDoc Amsterdam Interface + access to Orange internal ONAP lab: [http://onap.readthedocs.io/en/latest/submodules/so.git/docs/SO\_R1\_Interface.html#](http://onap.readthedocs.io/en/latest/submodules/so.git/docs/SO_R1_Interface.html)

## Service Catalog:

* Only characteristics at service level will be retrieved in ONAP Tosca file. For example if an ONAP service is composed of VNF and the VF module, the serviceSpecification resource will only feature characteristic describe in the ONAP service tosca model and not attributes in the tosca files for VNF or VF module.
* Only ‘basic’ service characteristics will be managed in this release. By ‘basic’ we mean string, boolean, integer parameter type and we do not manage ‘map’ or ‘list parameter type

## Service order

* ServiceOrder will manage only ‘add’ and ‘delete’ operation (no change).
* Management of ONAP customer for add service action:

With the current version of APIs used from SO and AAI we need to manage a ‘customer’. This customer concept is confusing with Customer BSS concept. We took the following rules to manage the ‘customer’ information:

* + **It could be provided through a serviceOrder** in the service Order a relatedParty with role ‘ONAPcustomer’ should be provided in the serviceOrder header (we will not consider in this release the party at item level); **External API component will check if** this customer **exists** **and create** it in AAI **if not**.
  + **If no relatedParty are provided the service** will be affected to ‘generic’ customer (dummy customer) – we assume this ‘generic’ customer always exists.
* **One** service order item will be map to **one or several orchestrated** request to SO but:
  + We assume SO provide **synchronous** response
  + We manage only SO request to trigger service (service instance) and optionally to VNF in this release (no VF Module triggering for example). This means that a BPMN must be defined to drive complete service fulfilment.
* Cloud & tenant information must be defined in the external API property file.

# Questions

|  |  |  |  |
| --- | --- | --- | --- |
| Category | Question | Answer | Comment |
| Catalog | Do SDC able to provide on the fly information by API? Able meaning can be used 24/7 to provide information |  | Linked to Catalog source choice. |
| Catalog | Do SDC able to provide on the fly information by API?  SO has runtime catalog information – is it complete?Do we find there the toscaUrl ? |  | Linked to catalog source choice |
| Catalog | [Where to find catalog information: SDC or SO? | SDC – To retrieve SDC service Template Tosca file + Source Amsterdam API | SDC has ‘complete’ information’. |
| Catalog | What are the difference between uuid and invariantUUID? And which one will be communicated to BSS? | Assumption: serviceSpec will be mapped to uuid. |  |
| Inventory | “*A service-instance is related to a customer and service-subscription in A&AI inventory (those objects are pre-created outside of MSO).*”  Is it required to create in A&AI customer and then service-subscription before to trigger service instance creation request in SO?  How we create this object – directly in A&AI? | Yes  Yes | We plan to use a ‘generic’ customer if it is not provided. |
| Orchestration | [**Critical**] Do we need to “Crack the Tosca”? meaning from a serviceSpec get the Tosca and understand from this file the orchestration of Service/VNF/VF Module to be created | No.  We’ll only manage service level and optional VNF level. No need to ‘crack’ the tosca to retrieve other resource like VF Module |  |
| Orchestration | If we consider that a service order item fulfillment in ONAP will be tackled through creation of service, VNF and VF Module, how the order item service characteristics are mapped to these 3 objects? | Assumption: For Beijing we restrict the scope to service and VNF |  |
| Orchestration/  Inventory | How complex characteristics object are described in ONAP? There are only name/value pattern? | First Service characteristic are defined in the Tosca model in *topology\_template* and then *inputs* |  |
| API Model | Do we remove on the API resource the classes/attributes not managed in this release (example appointment in the serviceOrder?) |  |  |
| Orchestration | How are managed service changes in ONAP? | Change operation is not managed in the Beijing release. |  |
| Orchestration | How are filled the CloudConfiguration attributes (cloudOwner, IcpCloudRegionId & tenantId)? | These 3 Attributes will have default values to be initialized in the external API component |  |

# Use-Case

## vFirewall

<https://wiki.onap.org/pages/viewpage.action?pageId=3246170>

Catalog description:

Service vFirewall SDC description (response)



This service is composed by 2 VNF.

VNF (Resource) SDC description:

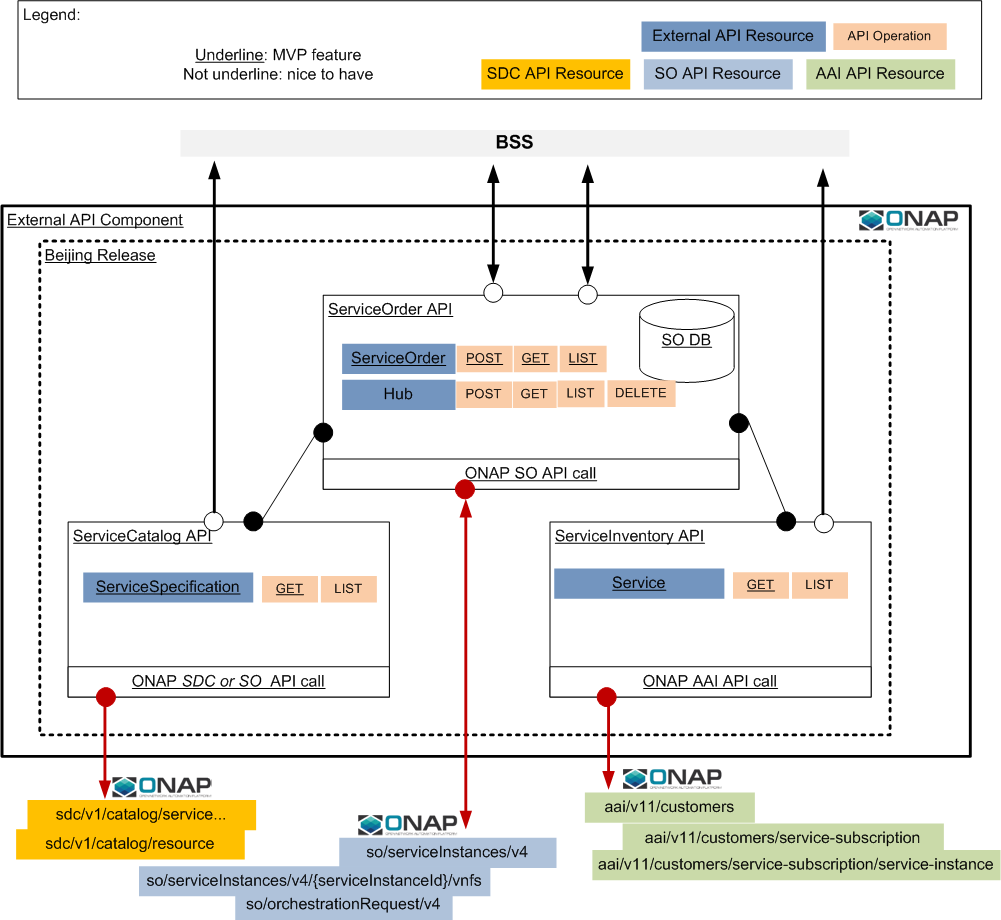




Tosca model for this Service vFirewall:



# Global Architecture



## External API – internal attributes

Following attributes are managed in the External API component to configure it:

|  |  |
| --- | --- |
| Name | Behavior |
| CallForVNF | A boolean – if set to **yes**, external API will trigger a SO call for service and then for VNF. If set to **no**, only service level call will be sent to SO.  By default ‘no’ |
| cloudOwner | Mandatory |
| lcpCloudRegionId | Mandatory (prerequisite: it exists and it is already related to the cloudOwner). |
| tenantId | Mandatory (prerequisite: it exists and related to the cloud region id). |

# TMF API

## TMF ServiceOrder API

### TMF Resource Model:



**ONAP Beijing restriction**

Following restrictions apply:

* ServiceOrder:
  + expectedCompletionDate will not be implemented
  + NotificationContact will not be implemented
  + requestedStartDate will not be managed
  + href will be managed **only** for service order resource (present in the GET and POST answer). Others href will not be managed
  + @schemaLocation will not be managed for serviceOrder class
  + Priority is not managed
* OrderRelationship
  + They will be stored but not managed (no dependency between SO managed)
* RelatedPartyRef
  + Assumption: If the serviceOrder feature a relatedParty at order level with role ‘ONAPcustomer’, we’ll use this one as customer in ONAP (first look for this one and create it if not exist). RelatedParty at order item are not considered to map with ONAP. If no relatedPartyRef with ‘ONAPcustomer’ role are provided we’ll use ‘generic’ customer in ONAP and add new service to this generic customer.
* Appointment
  + Not managed & not implemented in this release
* ServiceOrderItem:
  + **modify** action will **not** be **managed** – only add /noChange/delete will managed.
* Service
  + serviceState is optional but we will consider only ‘active’ request for an add action (not possible to add a new service in another state than active)
* OrderItemRelationship
  + They will be stored but only reliesOn type will be managed to orchestrate service order item fulfillment
* Place
  + Not managed & not implemented in this release
* ServiceRelationship
  + They will be stored but only reliesOn type will be managed to support service dependency.
  + API structure allows describing a graph of service via serviceRelationship but this capability will not be managed in this release; Only Ref is considered.
* Note
  + Not managed & not implemented in this release

### Swagger

TMF API swagger could be retrieved under TMF GitHub : <https://github.com/tmforum-apis/TMF641_ServiceOrder>

## ServiceInventory API

### Resource Model:



**ONAP Beijing restriction**

Following restrictions apply:

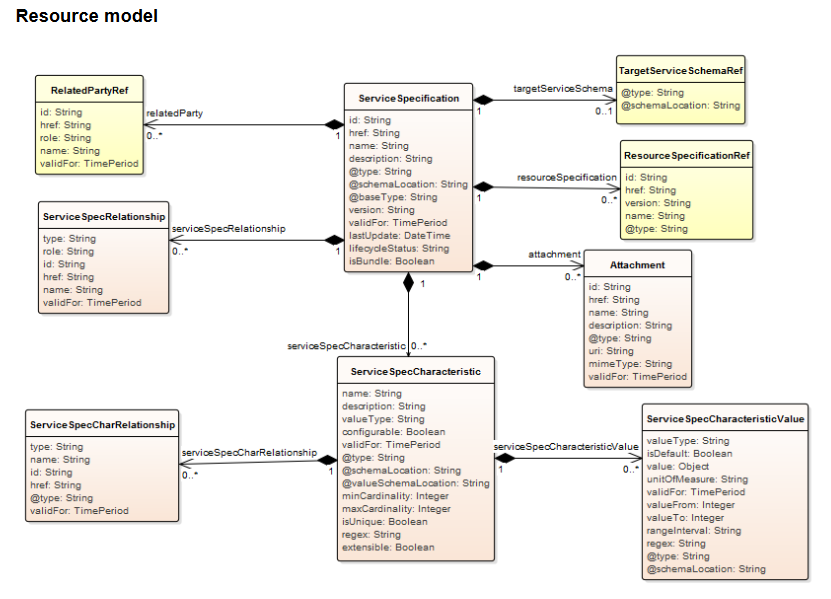
* Service:
  + Only following attributes will be retrieve in service inventory: *to be completed*
* Place will not retrieved
* Note will not be retrieved
* ServiceRelationship will not be retrieved
* ServiceOrderRef will not be retrieved
* SupportingService will not be retrieved

### Swagger

TMF API swagger could be retrieved under TMF GitHub: <https://github.com/tmforum-apis/TMF638_ServiceInventory>

## ServiceCatalog API

### Resource Model:



### Swagger

This API must be moved to public repository

<https://github.com/tmforum-apis/WIP_TMF633_ServiceCatalog>

# Service Catalog Mapping

## Sources

* <http://onap.readthedocs.io/en/latest/submodules/so.git/docs/SO_R1_Interface.html#get-specific-asset-detailed-metadata>
* <https://wiki.onap.org/display/DW/ExtAPI+Service+Catalog+Working+Space>

## API Description

### Swagger:

<https://app.swaggerhub.com/apis/bigludo7/ServiceCatalogManagement/0.1.8>

### Lifecycle:

Alignment to ONAP lifecycle (see table below)

### API Operation provided:

|  |  |  |
| --- | --- | --- |
| Operation on Entities | Uniform API Operation | Description |
| Query Entities | GET serviceSpecification/{id}  GET serviceSpecification | GET must be used to retrieve a (list of) representation of a service specification. |

### ServiceSpecification retrieval limitation:

Based on SDC limitations, only attributes category and distributionStatus are available for serviceSpecification filtering.

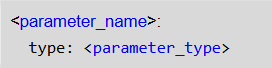
## Postman Collection

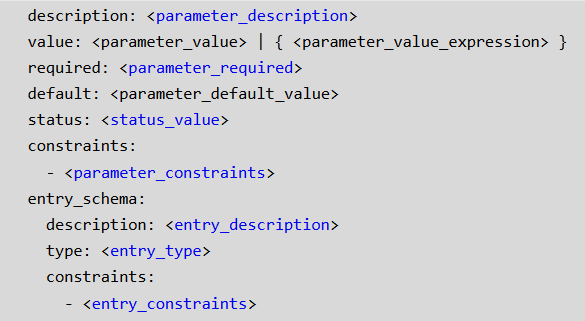


## Mapping rules

Based on SDC APIs with **assetType=services**:

* GET {{url}}/sdc/v1/catalog/services/?distributionStatus=DISTRIBUTED&category=Network Service to make a search on Service in SDC based on criteria – see SDC2 in postman
* GET {{url}}/sdc/v1/catalog/services/{id}/metadata to retrieve all information for one service – see SDC1 request in postman. Few codes to highlight:
  + - Code 200 is send if complete information are retrieved
    - Code 206 is send if partial information are retrieved (for example the Tosca file to retrieve service characteristic is missing or impossible to read)
    - Code 404 if service (id) did not exists.
* GET {{url}}/sdc/v1/catalog/services/3dd3923d-1681-4f5b-99bb-f695ab147004/toscaModel to retrieve the tosca czar file. Within this file package we’ll consider the following file: “service-{serviceName}-template.yml”.
* For the resource model customization name and id consider information under topology\_template/node\_template for each resource.
* For the characteristic consider in the Tosca file **topology\_template** section in the **inputs section:**





(<http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/os/TOSCA-Simple-Profile-YAML-v1.0-os.pdf> p105)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TMF | | | | | | ONAP | Find (2) | Implemented in Beijing |
| ServiceSpecification | | | | | |  | x | yes |
|  | id | | | | | uuid | x | yes |
|  | href | | | | | blank |  | yes |
|  | name | | | | | name | x | yes |
|  | description | | | | | description | x | yes |
|  | @type | | | | | valued “ONAPservice” |  | yes |
|  |  | invariantUUID | | | | invariantUUID | x | yes |
|  |  | toscaModelURL | | | | toscaModelURL |  | yes |
|  |  | toscaResourceName | | | | toscaResourceName |  | yes |
|  |  | category (1) | | | | category | x | yes |
|  |  | subcategory (1) | | | | subcategory |  | yes |
|  |  | distributionStatus | | | | distributionStatus – values are: DISTRIBUTION\_NOT\_APPROVED, DISTRIBUTION\_APPROVED, DISTRIBUTED, DISTRIBUTION\_REJECTED | x | yes |
|  | @schemaLocation | | | | | blank |  | yes |
|  | @baseType | | | | | blank |  | yes |
|  | version | | | | | version | x | yes |
|  | validFor | | | | | blank |  | no |
|  | lastUpdate | | | | | blank |  | no |
|  | lifecycleStatus | | | | | lifecycleState - Values are:NOT\_CERTIFIED\_CHECKOUT, NOT\_CERTIFIED\_CHECKIN, READY\_FOR\_CERTIFICATION, CERTIFICATION\_IN\_PROGRESS, CERTIFIED | x | yes |
|  | isBundle | | | | | Blank |  | no |
|  | targetServiceSchemaRef | | | | | not used |  | yes |
|  |  | | @type | | | not used |  | yes |
|  |  | | @schemaLocation | | | not used |  | yes |
|  | attachement | | | | | Proposal to map to ONAP artefact |  | yes |
|  |  | | id | | | artifactUUID |  | yes |
|  |  | | href | | | blank |  | no |
|  |  | | name | | | artifactName |  | yes |
|  |  | | description | | | artifactDescription |  | yes |
|  |  | | @type | | | valued “ONAPartifact” |  | yes |
|  |  | |  | artifactLabel | | artifactLabel |  | yes |
|  |  | |  | artifactGroupType | | artifactGroupType |  | yes |
|  |  | |  | artifactTimeout | | artifactTimeout |  | yes |
|  |  | |  | artifactChecksum | | artifactChecksum |  | yes |
|  |  | |  | artifactVersion | | artifactVersion |  | yes |
|  |  | |  | generatedFromUUID | | generatedFromUUID |  | yes |
|  |  | | url | | | artifactUrl |  | yes |
|  |  | | mimeType | | | artifactType |  | yes |
|  |  | | validFor | | | blank |  | no |
|  | relatedParty | | | | |  |  | yes |
|  |  | | id | | | lastUpdaterUserId | x | yes |
|  |  | | href | | | blank |  | no |
|  |  | | role | | | valued to “lastUpdater” | x | yes |
|  |  | | name | | | lastUpdatedFullName |  | yes |
|  |  | | validFor | | | blank |  | no |
|  | serviceSpecRelationship | | | | | No Mapping – No serviceSpec relationship described in ONAP?. |  | no |
|  |  | | … | | |  |  |  |
|  | resourceSpecification | | | | |  |  | yes |
|  |  | | id | | | resourceUUID |  | yes |
|  |  | | href | | | blank |  | no |
|  |  | | version | | | resourceVersion |  | yes |
|  |  | | name | | | resourceName |  | yes |
|  |  | | @type | | | valued “ONAPresource” |  | yes |
|  |  | |  | resourceInstanceName | | resourceInstanceName |  | yes |
|  |  | |  | resourceInvariantUUID | | resourceInvariantUUID |  | yes |
|  |  | |  | resourceType | | resou**cr**eType |  | yes |
|  |  | |  | modelCustomizationName | | (Tosca) name |  |  |
|  |  | |  | modelCustomizationId | | (Tosca) customizationUUID |  |  |
|  | serviceSpecCharacteristic | | | | | ServiceSpecCharacteristic are retrieved in the serviceTosca file in the **topology\_template** section in the **inputs section**. |  | yes |
|  |  | | name | | | parameter\_name |  | yes |
|  |  | | description | | | parameter\_description |  | yes |
|  |  | | valueType | | | parameter\_type |  | yes |
|  |  | | configurable | | |  |  | no |
|  |  | | validFor | | |  |  | no |
|  |  | | @type | | | valued “ONAPserviceCharacteristic” |  | yes |
|  |  | |  | required | | parameter\_required – if not fielded by default ‘true’ |  | yes |
|  |  | |  | status | | status\_value |  | yes |
|  |  | | @schemaLocation | | |  |  | yes |
|  |  | | @valueSchemaLocation | | |  |  | no |
|  |  | | minCardinality | | |  |  | no |
|  |  | | maxCardinality | | |  |  | no |
|  |  | | isUnique | | |  |  | no |
|  |  | | regex | | |  |  | no |
|  |  | | extensible | | |  |  | no |
|  |  | | serviceSpecCharRelationship | | | No mapping |  | no |
|  |  | |  | | … |  |  |  |
|  |  | | ServiceSpecCharacteristicValue | | | ServiceSpecCharacteristicValue are retrieved in the serviceTosca file |  | yes |
|  |  | |  | | valueType | parameter\_type. We do not manage parameter\_type= list or map for this release |  | yes |
|  |  | |  | | isDefault | Yes for the value designed under: |  | yes |
|  |  | |  | | value | One ServiceSpecCharacteristicValue per value.  Values are described in constraints/valid\_values |  | yes |
|  |  | |  | | unitOfMeasure |  |  | no |
|  |  | |  | | validFor |  |  | no |
|  |  | |  | | valueFrom |  |  | no |
|  |  | |  | | valueTo |  |  | no |
|  |  | |  | | rangeInterval |  |  | no |
|  |  | |  | | regex |  |  | no |
|  |  | |  | | @type |  |  | no |
|  |  | |  | | @schemaLocation |  |  | no |

1. Category and subcategory are mapped directly in the serviceSpecification resource attributes because we do not provide serviceCategory resource in Beijing release. Must be corrected in the future.
2. Only attributes flagged with a ‘x’ will be available in the getList API (GET with search criteria). All Attributes will be retrieved for a GET with id

# Service Order

## Sources

* <http://onap.readthedocs.io/en/latest/submodules/so.git/docs/SO_R1_Interface.html#north-bound-apis>

## API Description

### Swagger:

<https://app.swaggerhub.com/apis/bigludo7/ServiceOrderManagement/0.1.10>

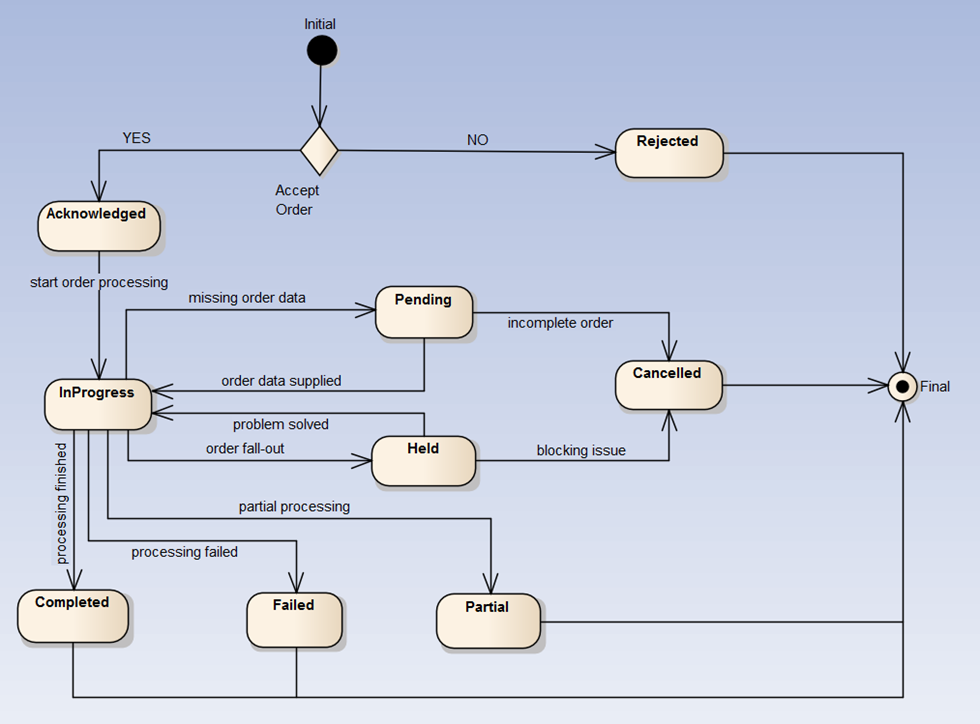
<https://app.swaggerhub.com/apis/bigludo7/EventManagement/1.0.0>

### Lifecycle:

Here is the state machine diagram for a Service order. Each order state is described in the table below.

The order item states are the same as the order ones except ‘Partial’ status which is not available for service order item.

Note that the order and order item states are tightly linked and need to be consistent:



**ONAP Beijing restriction**

Following restrictions apply:

* State management
  + **Only Acknowledged, Rejected, InProgress, Completed, Failed, Partial will be managed for service order state (Held, Pending and Cancelled are not managed)**
  + **Only Acknowledged, Rejected, InProgress, Completed, Failed will be managed for service order item state**
  + No mechanism to cancel order or order item is provided.
  + ‘Pending’ and ‘held’ which are used to manage temporary fulfilment stop are not managed in this version.

States are only managed by ServiceOrder component and could not be updated from north side via API. Accordingly to service order item fulfillment progress, order item state are updater. Order state is automatically updated based on item state – Following rules applied:

* + If action = noChange, orderItemState change to Completed then consider below table
  + Else consider following table

|  |  |  |
| --- | --- | --- |
| Event | service order Item state | service order state |
| A service order item has be fully delivered | change to ‘Competed’ | 1. All other order items are ‘Completed’ 🡪 order sate change to ‘Completed’ 2. All other order items are ‘completed’ and ‘failed’ (at least 1) 🡪 order sate change to ‘Partial |
| A service order item has be not deliverable because an definitive issue | change to ‘Failed | 1. All other order items are ‘Failed 🡪 order sate change to ‘Failed 2. All other order items are ‘Completed’ (at least 1) and ‘Failed’ 🡪 order sate change to ‘Partial |
| A service order item fulfilment starts in ONAP SO | change to ‘inProgress’ | change to ‘inProgress’ (if not yet) |
| A service order item fails a consistency check (ex: serviceSpecification id provided is not existing) | change to ‘Rejected’ | change to ‘Rejected’ |
| A new service order is received | all order item are initialized with ‘acknowledged’ state | If all order item have ‘acknowledged’ state, order has ‘acknowledged’ state |

### API Operation provided:

serviceOrder resource:

|  |  |  |
| --- | --- | --- |
| Operation on Entities | Uniform API Operation | Description |
| Query Entities | GET serviceOrder/{id}  GET serviceOrder | GET must be used to retrieve a (list of) representation of a resource. |
| Create Entity | POST serviceOrder | POST must be used to create a new resource |
| Partial Update of an Entity  (Not provided in Release 1) | PATCH serviceOrder | PATCH must be used to partially update a resource |
| Delete Entity  Only use for ADMIN.  Provided only for testing purpose. | DELETE serviceOrder/{id} | DELETE must be used to delete a resource |

Hub resource:

|  |  |  |
| --- | --- | --- |
| Operation on Entities | Uniform API Operation | Description |
| Query Entities | GET hub/{id}  GET hub | GET must be used to retrieve a (list of) representation of a resource. |
| Create Entity | POST hub | POST must be used to create a new resource |
| Delete Entity | DELETE hub | DELETE must be used to delete an existing resource |
| Partial Update of an Entity  (Not provided in Release 1) | PATCH hub | PATCH must be used to partially update a resource |

## Postman Collection:



## POST Service Order High Level view

Following diagram illustrates steps to be processed:



## Detailed view step by step

**Check Order**

* Payload compliance (Cardinality, required attribute, enumList)
* If orderItem.action = ‘add’ then
  + orderItem.service.serviceSpecification.id must be provided else 422/100 error code and the order is not stored
* If orderItem.action <> ‘add’ then
  + orderItem.service.id and orderItem.service.serviceSpecification.id must be provided else 422/101 error code and the order is not stored (Note: servicespecification.id is useful to find service instance in A&AI)
* **Got to Store Order**

**Store Order**

* When the service order is stored, following attribute are valued by the SO component:
  + id
  + href for the service order class
  + state (set to ‘Acknowledged’) //*Notification send if subscribed* //
  + orderDate: timestamp
  + orderItem.state (set to ‘Acknowledged’)
* Store order
* Return 201 code to POST requester + provide serviceOrder representation
* Got to **Check Order Consistence**

**Check Order Consistence**

For each order item:

* **If orderItem.action = add** then
  + Check service specification existence in the catalog 🡪 use internal API GET /serviceCatalog/serviceSpecification/{id}
    - If response distinct from success (200) then the order item state must shift to ‘Rejected’. (error code 422/102) [*retrieved data must be temporarily stored because useful in following steps*]
    - If response 200 (success) – store service name and **go to next item**.
  + If service.id is filled the order item state must shift to ‘Rejected’. (error code 422/103)
  + Check tenant/cloud information consistency.
* If orderItem.action <> add then
  + if a relatedParty at order level – role: “ONAPcustomer” is provided
    - Check customer existence in the inventory () 🡪 use A&AI api GET {{url}}/aai/v11/business/customers/customer/{relatedParty.id}
    - if AAI response distinct from success (200) the order item state must shift to ‘Rejected’ (422/104)
  + Else we will consider the service to be modified is under ‘generic’ customer.
  + Check service existence in the inventory🡪 use internal API GET /serviceInventory/service/{id}
    - if response distinct from success (200) the order item state must shit to ‘Rejected’. (422/105)

Once all order item(s) proceed:

* If not item shifted to ‘Rejected’
  + If all order items have action distinct to ‘add’ then go to **Send order Item to SO** for each order item
  + Else if all items action are ‘add’ then go to **Create A&AI customer**
  + Else trigger **Create A&AI serviceType**
* Else
  + Order state must be shift to ‘Rejected’ and timestamp completionDate with current dateTime - End of procession for this service ordering *//Notification send if subscribed //*

**Create A&AI customer**

To be done only if **all action item**(s) are ‘add’

* if a relatedParty at order level – role: “ONAPcustomer” is provided
  + Check customer existence in the inventory (Request provided in postman AAI1) 🡪 use A&AI api GET {{url}}/aai/v11/business/customers/customer/relatedParty.id–
  + if AAI response distinct from success (see POSTMAN AA1bis) the customer must be created in A&AI PUT {{url}}/aai/v11/business/customers/customer/relatedParty.id for role ‘ONAPcustomer’}. Customer creation is illustrated in postman AAI2
    - global-customer-id = relatedParty.id for role=”ONAPcustomer”
    - subscriber-name = relatedParty.name for role=”ONAPcustomer ”or ‘genericName if not provided
    - subscriber-type = ‘BSS’
* Else we will consider the service to be modified is under ‘generic’ customer.

Go to **Create A&AI service type**

**Create A&AI service type**

**To be done for all order item with action add**.

* First it is necessary to check if the service-type already exists under the customer. The service-type is filled with service name (retrieved in the SDC call in CheckOrder consistencies step. In order to do that we use the operation GET {{url}}/aai/v11/business/customers/customer/{relatedParty.id or generic}/service-subscriptions– this is described in postman AAI3
* If not it must be created – use A&AI API PUT {{url}}/aai/v11/business/customers/customer/{relatedParty.id or generic}/service-subscriptions/service-subscription/{service.name} – this is described in postman AAI12
* If yes, go to next item

Once all item processed go to **Send order Item to SO**

**Send order Item to SO**

POSTMAN Collection



* When fulfillment starts for the first item of the order, timestamp startDate with current date time.
* If orderItem **action** is ‘**add’** and orderItemState is ‘Acknowledged’
  1. Check if this item is dependent to other item - This order item has orderItemRelationship with type reliesOn?
     1. If not go to **b**.
     2. If yes, check targeted order item state
        1. If ‘Completed’ go to **b**.
        2. Else – this order item could not be processed at this time
  2. Trigger SO APIs
     1. For service use POST {{url}}/ecomp/mso/infra/serviceInstances/v4 – This is illustrated in SO1, SO3 and SO5 examples (all the same)

|  |  |  |  |
| --- | --- | --- | --- |
| **ONAP API** | | | **Mapping** |
| modelInfo | | |  |
|  | ModelType | | “service” |
|  | ModelnvariantId | | invariantUUID (retrieved from SDC Call) |
|  | ModelNameVersionId | | serviceSpecification.id |
|  | ModelName | | name(retrieved from SDC Call) |
|  | ModelVersion | | version (retrieved from SDC Call) |
|  | ModelCustomizationName | |  |
| subscriberInfo | | |  |
|  | GlocalSubscriberId | | relatedParty.id for role ‘ONAPcustomer’ if provided else id of the ‘generic’ customer |
|  | SubscriberName | | relatedParty.id for role ‘ONAPcustomer’ if provided else id of the ‘generic’ customer |
| requestInfo | | |  |
|  | InstanceName | | service.name |
|  | ProductFamilyId | |  |
|  | Source | | Value to ‘VID’ |
|  | SupressRollback | | false |
|  | requestorId | | Value to ‘ExternalAPI’ |
| requestParameters | | |  |
|  | SubscriptionServiceType | | service-type created in AAI (correspond to service name retrieved in SDC) |
|  | aLaCarte | | true |
|  | autoBuildVfModules | |  |
|  | cascadeDelete | |  |
|  | usePreload | | Yes |
|  | rebuildVolumeGroups | |  |
|  | UserParams | | direct mapping to characteristic provided in the serviceOrder (serviceOrderItem.service.serviceCharacteristic) |
|  |  | name | name |
|  |  | value | value.serviceCharacteristicValue |
|  | cloudConfiguration | | Default values defined in External API parameters |
|  |  | lcpCloudRegionId | lcpCloudRegionId |
|  |  | tenantId | tenantId |

* + - * If return code is different to 201
        + change orderItem state to ‘Failed’
        + Trigger **Update service order**
      * Else retrieve in the response
        + instanceId – this id must be stored in the following attribute : serviceOrder.serviceOrderItem.service.id (for the order item processed)
        + requestId – this id must be stored in the external API at the level of the serviceOrderItem because its corresponds to the ONAP/SO request ID for the service performed.
      * Check Request status via GET {{url}}/ecomp/mso/infra/orchestrationRequests/v4/{requestId}

Check requestStatus: it must be equal to (if percentProgress not equal to 100, we’ll pooling SO till percentProgress = 100):

* + - * If requestState <> “COMPLETE” and percentProgress = 100, we assume an error occurred:
        + change orderItem state to ‘Failed’
        + Trigger **Update service order**
    1. If CallForVNF is set to ‘no’ or if this service has no VNF to instantiate:
       - change orderItem state to ‘Completed’ - ‘Send order Item to SO’ step is finish for this item
       - Trigger **Update service order**
    2. Else… If CallForVNF is set to ‘yes’ - For each VNF

Use POST {{url}}/ecomp/mso/infra/serviceInstances/v4/{serviceInstance.id}/vnfs - – The {serviceInstance.id} is retrieved from i. The VNF to be created are described in the service description retrieved from the tosca file retrieved SDC. One VNF must be created for each resources defined with type resourceType= “VF”. This is illustrated in SO7 and SO9 examples (on 2 distinct vnf).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ONAP API** | | | | | **Mapping** |
| modelInfo | | | | |  |
|  | ModelType | | | | “vnf” |
|  | ModelnvariantId | | | | resources.resourceInvariantUUID |
|  | ModelNameVersionId | | | | resources.resourceUUID |
|  | ModelName | | | | resources.resourceName |
|  | ModelVersion | | | | resources.resourceVersion |
|  | ModelCustomizationName | | | | Retrieved in the tosca file: section topology\_template/node\_template (type must be equal to VF) – use name |
|  | ModelCustomizationId | | | | Retrieved in the tosca file: section topology\_template/node\_template (type must be equal to VF) – use customizationUUID |
| requestInfo | | | | |  |
|  | InstanceName | | | | Must be automatically attributed from external ID component – concatenate serviceInstance.name+’vnf’+ sorting number (01, 02, 03, etc…) |
|  | Source | | | | Value to ‘VID’ |
|  | SupressRollback | | | | false |
|  | requestorId | | | | externalAPI |
|  | productFamilyId | | | | blank |
| requestParameters | | | | |  |
|  | UserParams | | | | direct mapping to characteristic provided in the serviceOrder (serviceOrderItem.service.serviceCharacteristic) |
|  |  | name | | | …name |
|  |  | value | | | value.serviceCharacteristicValue |
|  | cloudConfiguration | | | | Default values defined in External API parameters |
|  |  | lcpCloudRegionId | | | lcpCloudRegionId |
|  |  | tenantId | | | tenantId |
|  | relatedInstanceList | | | |  |
|  |  | relatedInstance | | |  |
|  |  |  | instanceId | | must be filled with the service instance id retrieve from the service request in step i |
|  |  |  | modelInfo | | These attributes are retrieved for the service from where the VNF is created. This is catalog information |
|  |  |  |  | modelType | “service” |
|  |  |  |  | modelName | name |
|  |  |  |  | modelInvariantId | invariantUUID |
|  |  |  |  | modelVersion | version |
|  |  |  |  | modelNameVersionId | id |

* + - If return code is different to 201
      * change orderItem state to ‘Failed’
      * delete other VNF created (if any) (described in DELETE action)
      * delete service created (described in DELETE action)
      * Trigger **Update service order**
    - Else retrieve in the response
      * + instanceId – no need to be stored in externalAPI component
        + requestId – this id must be stored in the external API at the level of the serviceOrderItem because its corresponds to the ONAP/SO request ID for the vnf performed. That means that if a service in ONAP has 2 vnf, the corresponding serviceOrder item will be related to 3 ONAP/SO requestId (the one for the ONAP service and 2 for the vnf).
      * Check Request status via GET {{url}}/ecomp/mso/infra/orchestrationRequests/v4/{requestId}

Check requestStatus: it must be equal to (if percentProgress not equal to 100, we’ll pooling SO till percentProgress = 100):

* + - * If requestState <> “COMPLETE” and percentProgress = 100 for at least one vnf, we assume an error occurred:
        + change orderItem state to ‘Failed’
        + delete other VNF created (if any) (described in DELETE action)
        + delete service created (described in DELETE action)
        + Trigger **Update service order**
      * Else when requestState = ‘COMPLETE’ for **all** vnf
        + change orderItem state to ‘Completed’ - ‘Send order Item to SO’ step is finish for this item
        + Trigger **Update service order**
* If orderItem is ‘no-change’ and orderItemState is ‘Acknowledged’
  + change orderItem state to ‘Completed’
  + Trigger **Update service order**
* If orderItem is ‘delete’ and orderItemState is ‘Acknowledged’
* We need first to:
  + Retrieve catalog information (from serviceSpecificationId)
  + Retrieve inventory information (from customer, serviceSpecificationName, serviceId)
* If CallForVNF is set to ‘yes’ it means that we have first to retrieve and delete VNF before to delete the service. If CallForVNF is set to ‘no’ go directly to **DeleteService** below.

**DeleteVNF**:

* The list of vnf to be deleted has been retrieved in the previous inventory call
* The vnf catalog information has been retieved in the previous catalog call
* We use SO API: DELETE {{url}}/ecomp/mso/infra/serviceInstances/{service.id}/vnfs/{supportingResource.id} The supportingResource.id has been retrieved from the inventory call. Body must be filled as described below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ONAP API** | | | | | **Mapping** |
| modelInfo | | | | |  |
|  | ModelType | | | | “vnf” |
|  | ModelnvariantId | | | | *Catalog* resourceSpecification.resourceInvariantUUID |
|  | ModelNameVersionId | | | | *Catalog* resourceSpecification.resourceUUID |
|  | ModelName | | | | *Catalog* resourceSpecification.name |
|  | ModelVersion | | | | *Catalog* resourceSpecification.version |
|  | ModelCustomizationName | | | | *Catalog* resourceSpecification. modelCustomizationName |
|  | ModelCustomizationId | | | | *Catalog* resourceSpecification.modelCustomizationUUID |
| requestInfo | | | | |  |
|  | InstanceName | | | | *Inventory* supportingResource.name |
|  | Source | | | | Value to ‘VID’ |
|  | SupressRollback | | | | false |
|  | requestorId | | | | externalAPI |
|  | productFamilyId | | | | blank |
| requestParameters | | | | |  |
|  | UserParams | | | |  |
|  |  | name | | |  |
|  |  | value | | |  |
|  | cloudConfiguration | | | | Default values defined in External API parameters |
|  |  | lcpCloudRegionId | | | lcpCloudRegionId |
|  |  | tenantId | | | tenantId |
|  | relatedInstanceList | | | |  |
|  |  | relatedInstance | | |  |
|  |  |  | instanceId | | must be filled with the service instance id i |
|  |  |  | modelInfo | | These attributes are retrieved for the service from where the VNF is created. This is catalog information |
|  |  |  |  | modelType | “service” |
|  |  |  |  | modelName | *Catalog* name |
|  |  |  |  | modelInvariantId | *Catalog* invariantUUID |
|  |  |  |  | modelVersion | *Catalog* version |
|  |  |  |  | modelNameVersionId | *Catalog* id |

* If return code is different to 201
  + - * If previously VNF have been deleted, the must be re-created (same request than the one used for ADD)
      * change orderItem state to ‘Failed’
* Trigger **Update service order**
  + **Else** retrieve in the response
    - * instanceId – no need to be stored in externalAPI component
      * requestId – this id must be stored in the external API at the level of the serviceOrderItem because its corresponds to the ONAP/SO request ID for the vnf performed.
      * Check Request status via GET {{url}}/ecomp/mso/infra/orchestrationRequests/v4/{requestId}

Check requestStatus: it must be equal to (if percentProgress not equal to 100, we’ll pooling SO till percentProgress = 100):

* + - * If requestState <> “COMPLETE” and percentProgress = 100 for at least one vnf, we assume an error occurred:
        + change orderItem state to ‘Failed’
        + If previously VNF have been deleted, the must be re-created (same request than the one used for ADD)
        + Trigger **Update service order**
      * Else when requestState = ‘COMPLETE’ for **all** vnf deletion
        + Go to DeleteService

**DeleteService**:

* We use SO API: DELETE {{url}}/ecomp/mso/infra/serviceInstances/{service.id}. Body must be filled as described below:

|  |  |  |  |
| --- | --- | --- | --- |
| **ONAP API** | | | **Mapping** |
| modelInfo | | |  |
|  | ModelType | | “service” |
|  | ModelnvariantId | | invariantUUID (retrieved from SDC Call) |
|  | ModelNameVersionId | | serviceSpecification.id |
|  | ModelName | | name(retrieved from SDC Call) |
|  | ModelVersion | | version (retrieved from SDC Call) |
|  | ModelCustomizationName | |  |
| subscriberInfo | | |  |
|  | GlocalSubscriberId | | relatedParty.id for role ‘ONAPcustomer’ if provided else id of the ‘generic’ customer |
|  | SubscriberName | | relatedParty.id for role ‘ONAPcustomer’ if provided else id of the ‘generic’ customer |
| requestInfo | | |  |
|  | InstanceName | | *inventory* service.name |
|  | ProductFamilyId | |  |
|  | Source | | Value to ‘VID’ |
|  | SupressRollback | | false |
|  | requestorId | | Value to ‘ExternalAPI’ |
| requestParameters | | |  |
|  | SubscriptionServiceType | | *inventory* servicespecification.name |
|  | aLaCarte | | true |
|  | autoBuildVfModules | |  |
|  | cascadeDelete | |  |
|  | usePreload | | Yes |
|  | rebuildVolumeGroups | |  |
|  | UserParams | |  |
|  |  | name |  |
|  |  | value |  |
|  | cloudConfiguration | | Default values defined in External API parameters |
|  |  | lcpCloudRegionId | lcpCloudRegionId |
|  |  | tenantId | tenantId |

* If return code is different to 201
  + - * If previously VNF have been deleted, the must be re-created (same request than the one used for ADD)
      * change orderItem state to ‘Failed’
* Trigger **Update service order**
  + Else retrieve in the response
    - * instanceId – no need to be stored in externalAPI component
      * requestId – this id must be stored in the external API at the level of the serviceOrderItem because its corresponds to the ONAP/SO request ID.
      * Check Request status via GET {{url}}/ecomp/mso/infra/orchestrationRequests/v4/{requestId}

Check requestStatus: it must be equal to (if percentProgress not equal to 100, we’ll pooling SO till percentProgress = 100):

* + - * If requestState <> “COMPLETE” and percentProgress = 100 for at least one vnf, we assume an error occurred:
        + change orderItem state to ‘Failed’
        + If previously VNF have been deleted, the must be re-created (same request than the one used for ADD)
        + Trigger **Update service order**
      * Else when requestState = ‘COMPLETE’
        + change orderItem state to ‘Completed’ - ‘Send order Item to SO’ step is finish for this item
        + Trigger **Update service order**

**Check SO Fulfilment**

**ONAP Beijing restriction**

Not managed in this release – once all SO API triggered and got successful response we consider thee service order item as ‘Completed’ (not A&AI Check done)

**Update service order**

* Update serviceOrder state accordingly to service order item status (see table above in service order API lifecycle §)
* If order state changed to Completed or Partial timestamp completionDate with current dateTime

### Retrieve ServiceOrder

The operation GET serviceOrder without id allows querying External API to retrieve existing serviceOrder.

For this release, only a subset of criteria is available:

* externalId
* state
* description
* orderDate.gt (orderDate must be greater – *after* -than)
* orderDate.lt (orderDate must be lower-*before* - than)
* fields – attribute used to filter retrieved attributes (if needed) and also for sorted SO
* offset and limit are used for pagination purpose

As of now fields, offset and limit capabilities are tagged to nice to have.

We provide as well an operation GET with id – this operation will allow retrieving the full service order representation.

### ServiceOrder Notification

Only one notification will be managed for this release: 'ServiceOrderStateChangeNotification'

This allows triggering notification when the serviceOrder switch to the following order state: acknowledged, InProgress, completed, failed and partial.

To get notification, client app has to use HUB resource.

Example:

Client wants to subscribe to serviceOrder notification:

POST {api\_url}/HUB

Accept: application/json

{"callback": <http://in.listener.com>,

“query”:”eventType = 'ServiceOrderStateChangeNotification'

}

<http://in.listener.com>: url where client wishes to receive notification.

Seller will POST an Event to the Buyer:

POST {callback}/event

{

"eventType": "ServiceOrderStateChangeNotification",

"eventTime": "2017-09-27T05:46:25.0Z",

"eventId": "92445",

"event":

{

"serviceOrder": {

“id”: “456987”,

“href”: “http://www.....”,

“state”: “completed”

}

}

We’re expecting a standard HTTP 201 if event received.

# Service Inventory Mapping

## Sources

Internal tests on Orange Platform

## API Description

### Swagger

<https://app.swaggerhub.com/apis/bigludo7/ServiceInventoryManagement/0.1.6>

### Lifecycle:

No product inventory lifecycle provided in this version.

### API Operation provided:

|  |  |  |
| --- | --- | --- |
| Operation on Entities | Uniform API Operation | Description |
| Query Entities | GET service/{id}  GET service | GET must be used to retrieve a (list of) representation of a resource. |

*Note about DELETE operation: initially plan was to provide a DELETE service operation. This feature has been replaced and DELETE a service is motorized through a ServiceOrder/serviceOrderItem with action* DELETE.

### GET Service Inventory (list)

The GET (by list) has following parameter:

|  |  |
| --- | --- |
| id | id of the service instance (inventory) |
| serviceSpecification.id | id of the service specification (catalog) |
| serviceSpecification.name | name of the service specification (catalog) |
| relatedParty.id | if not filled we use ‘generic’ customer |

The API will not send a 404 error if this id did not exist but instead will send an empty list.

The GET response will be:

|  |  |  |  |
| --- | --- | --- | --- |
| TMF | | | ONAP |
| Service | | |  |
|  | id | | service-instance-id |
|  | name | | service-instance-name |
|  | serviceSpecification | |  |
|  | id | | model-version-id |
|  | name | | service-type |
|  | relatedParty | |  |
|  |  | id | from the input parameter or ’generic’ |
|  |  | role | “ONAPcustomer” |

1. If a request is send without any parameter, we’ll retrieve the list of service-instance for the ‘generic’ customer:
   * we use GET {{url}}/aai/v11/business/customers/customer/generic/service-subscriptions/ Operation to retrieve a list of service-type
   * and then for each service type we retrieve all service instance: GET {{url}}/aai/v11/business/customers/customer/{ generic}/service-subscriptions/service-subscription/{service-type}/service-instances/ Operation
2. If only customer parameter is filled (relatedParty.id + role= relatedParty’ONAPcustomer’) we’ll retrieve the list of service-instance for this customer (corresponding to serviceSpecification;name)– Same process than 1 except that customerId is used in place of ‘generic’ customer
3. If serviceSpecification.id or name is filled we’ll retrieve the list of Service instance (from this service specification) – We’ll use the customer id if provided (with Role=’ONAPcustomer) or generic if no customer id provided.
   * From the specification.id retrieve the specification.name (if not provided) 🡪use Catalog API
   * Then use GET {{url}}/aai/v11/business/customers/customer/{customerId or generic}/service-subscriptions/service-subscription/{serviceSpecification.name}/service-instances/ Operation

We do not provide any other capability in this release.

### GET Product Inventory by id

Because of AAI capability, additionally to the service id, customer id and [serviceSpecification.id or serviceSpecification.name] must be supplied. If the customer id is not supplied, External API will use ‘generic’ customer. If only specification.id is provided, it is required to retrieve the specification.name (use Catalog API).

The request to use is AAI API: GET {{url}}/aai/v11/business/customers/customer/{customerId or generic}/service-subscriptions/service-subscription/{servicespecification.name }/service-instances/service-instance/{service.id}.

#### Mapping table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TMF | | | | ONAP | Implemented in Beijing |
| Service | | | |  |  |
|  | id | | | service-instance-id | yes |
|  | href | | | blank | yes |
|  | name | | | service-instance-name | yes |
|  | type | | | “service-instance” | yes |
|  | description | | |  | no |
|  | state | | | blank | yes |
|  | category | | |  | no |
|  | isServiceEnabled | | |  | no |
|  | hasStarted | | | Valued to ‘yes’ | yes |
|  | startMode | | |  | no |
|  | isStateful | | |  | no |
|  | serviceDate | | |  | no |
|  | startDate | | |  | no |
|  | endDate | | |  | no |
|  | @type | | | “serviceONAP” | yes |
|  | @baseType | | | blank | yes |
|  | @schemaLocation | | | blank | yes |
|  | serviceSpecification | | |  | yes |
|  | id | | | model-version-id | yes |
|  | href | | | blank | yes |
|  | name | | | From the input parameter | yes |
|  | version | | | blank | yes |
|  | @type | | | valued “ONAPservice” | yes |
|  | @schemaLocation | | | blank | no |
|  | invariantUUID | | | model-invariant-id | yes |
|  | targetServiceSchemaRef | | |  | no |
|  |  | @type | |  | no |
|  |  | @schemaLocation | |  | no |
|  | characteristic | | |  | yes |
|  |  | name | |  | yes |
|  |  | valueType | | Valued to ‘string’ | yes |
|  |  | value | |  | yes |
|  |  | @type | | blank | yes |
|  |  | @schemaLocation | | blank | yes |
|  |  | serviceCharacteristicValue | |  | yes |
|  | serviceRelationship | | |  | no |
|  |  | … | |  |  |
|  | supportingService | | |  | no |
|  |  | … | |  |  |
|  | supportingResource | | | A supportingResource will be retrieved for each relationship of the relationship-list where related-link describe a vnf: "/aai/v11/network/**generic-vnfs/generic-vnf**/{vnf id}” – it will be necessary to call for each vnf the API GET {{url}}/aai/v11/network/generic-vnfs/generic-vnf/{*vnf id*} | yes |
|  | id | | | vnf-id |  |
|  | href | | | related-link |  |
|  | role | | | blank |  |
|  | name | | | vnf-name |  |
|  | @referredType | | | values to ‘ONAP resource’ |  |
|  | @schemaLocation | | | blank |  |
|  | status | | | prov-status |  |
|  | modelInvariantId | | | model-invariant-id |  |
|  | modelVersionId | | | model-version-id |  |
|  | modelCustomisationId | | | model-customisation-id |  |
|  | relatedParty | | |  | yes |
|  |  | id | | From the input parameter (customer.id) | yes |
|  |  | href | | blank | yes |
|  |  | role | | “ONAPcustomer” | yes |
|  |  | name | |  | no |
|  |  | @referredType | | blank | no |
|  |  | validFor | |  | no |
|  |  |  | startDateTime |  | no |
|  |  |  | endDateTime |  | no |
|  | serviceOrder | | |  | no |
|  |  | id | |  | no |
|  |  | href | |  | no |
|  |  | serviceOrderItem | |  | no |
|  | place | | |  | no |
|  |  | … | |  |  |
|  | note | | |  | no |
|  |  | … | |  |  |

# ONAP API Description

## SDC API

### API resource model:

Sources:

<http://onap.readthedocs.io/en/latest/submodules/so.git/docs/SO_R1_Interface.html#get-specific-asset-detailed-metadata>

<https://wiki.onap.org/display/DW/ExtAPI+Service+Catalog+Working+Space>

|  |  |
| --- | --- |
| Get List of Existing Catalog Assets | sdc/v1/catalog/{assetType} |
| Get Specific Asset Detailed Metadata | /sdc/v1/catalog/{assetType}/{uuid}/metadata |



## SO API

### API resource model:



## A&AI API

### API resource model:

