ONAP SDK

Release 9.2.1

Sylvain Desbureaux

Sep 27, 2022

CONTENTS

CHAPTER

ONE

DESCRIPTION

ONAP SDK is a client library written in Python for building applications to work with ONAP. The project aims to provide a consistent and complete set of interactions with ONAP's many services, along with complete documentation, examples, and tools.

Using few python commands, you should be able to onboard, distribute models and instantiate xNFs.

First beta release deals with ONAP "Legacy" APIs but new APIs, CDS and policy integration is planned for next releases.

CHAPTER

TWO

INTRODUCTION

It *should* be simple to use. Once you have installed the Python module, few lines of code are needed to onboard a Service:

```
from onapsdk.vf import Vf
from onapsdk.service import Service

# We assume here that the VF has been already onboarded
vf = VF(name="myVF")
service = Service(name="myService", resources=[vf])
service.onboard()
```

CHAPTER

THREE

INSTALLATION

3.1 Installing with pip

```
$ pip install onapsdk
```

3.2 Customize the configuration

You can customize the global settings of onapsdk by creating an environment variable ONAP_PYTHON_SDK_SETTINGS and a file my_settings.py.

By default the global settings are retrieved from the file located in src/onapsdk/configuration/global_settings.py. You can create your own customized configuration file and reference it through the environement variable. You can thus copy/paste the existing global_settings.py file, rename it as my_settings.py, adapt it with your favorite editor and export the environnement variable accordingly.

It can be useful to move from a nodeport to an an ingress based configuration or test different API versions.

```
$ export ONAP_PYTHON_SDK_SETTINGS="onapsdk.configuration.my_settings"
```

CHAPTER

FOUR

USAGE

A minimum knowledge of ONAP is needed, especially on the onboarding part.

4.1 Cloud configuration

4.1.1 Create a complex

```
from onapsdk.aai.cloud_infrastructure import Complex
cmplx = Complex.create(
    physical_location_id=COMPLEX_PHYSICAL_LOCATION_ID,
    data_center_code=COMPLEX_DATA_CENTER_CODE,
    name=COMPLEX_PHYSICAL_LOCATION_ID
)
```

4.1.2 Create cloud region

```
from onapsdk.aai.cloud_infrastructure import CloudRegion
cloud_region = CloudRegion.create(
    cloud_owner=CLOUD_OWNER,
    cloud_region_id=CLOUD_REGION,
    orchestration_disabled=False,
    in_maint=False,
    cloud_type=CLOUD_TYPE,
    cloud_region_version=CLOUD_REGION_VERSION
)
```

4.1.3 Link cloud region to complex

```
from onapsdk.aai.cloud_infrastructure import CloudRegion, Complex
# We assume that complex has been already created
cmplx = Complex(
    physical_location_id=COMPLEX_PHYSICAL_LOCATION_ID,
    name=COMPLEX_PHYSICAL_LOCATION_ID
)
cloud_region = CloudRegion.create(
    cloud_owner=CLOUD_OWNER,
    cloud_region_id=CLOUD_REGION,
    orchestration_disabled=False,
```

(continues on next page)

(continued from previous page)

```
    in_maint=False,
    cloud_type=CLOUD_TYPE,
    cloud_region_version=CLOUD_REGION_VERSION
)
cloud_region.link_to_complex(cmplx)
```

4.1.4 Add ESR Info to cloud region

```
from uuid import uuid4
from onapsdk.aai.cloud_infrastructure import CloudRegion
# We assume that cloud region has been already created
cloud_region = CloudRegion.get_by_id(
    cloud_owner=CLOUD_OWNER,
    cloud_region_id=CLOUD_REGION
)
cloud_region.add_esr_system_info(
    esr_system_info_id=str(uuid4()),
    user_name=VIM_USERNAME,
    password=VIM_PASSWORD,
    system_type=CLOUD_TYPE,
    service_url=VIM_SERVICE_URL,
    cloud_domain=CLOUD_DOMAIN
)
```

4.1.5 Register cloud to MultiCloud

```
from uuid import uuid4
from onapsdk.aai.cloud_infrastructure import CloudRegion
# We assume that cloud region has been already created
cloud_region = CloudRegion.get_by_id(
    cloud_owner=CLOUD_OWNER,
    cloud_region_id=CLOUD_REGION
)
cloud_region.add_esr_system_info(
    esr_system_info_id=str(uuid4()),
    user_name=VIM_USERNAME,
    password=VIM_PASSWORD,
    system_type=CLOUD_TYPE,
    service_url=VIM_SERVICE_URL,
    cloud_domain=CLOUD_DOMAIN
)
cloud_region.register_to_multicloud()
```

4.1.6 Get cloud region tenant

```
# We assume that cloud region has been already created
# and connected to multicloud
from onapsdk.aai.cloud_infrastructure import CloudRegion
cloud_region = CloudRegion.get_by_id(
    cloud_owner=CLOUD_OWNER,
    cloud_region_id=CLOUD_REGION
)
try:
    tenant = next(cloud_region.tenant)
except StopIteration
    # No Tenant found in cloud region
```

4.1.7 Create customer

```
from onapsdk.aai.business import Customer
customer = Customer.create(GLOBAL_CUSTOMER_ID, GLOBAL_CUSTOMER_ID, "INFRA")
```

4.1.8 Create customer service subscription

```
# We assume here that the service has been already onboarded
# and customer created
from onapsdk.aai.business import Customer

customer = Customer.get_by_global_customer_id(GLOBAL_CUSTOMER_ID)
customer.subscribe_service("service_type")

# Service subscriptions can be also created during Customer
# creation
from onapsdk.aai.business import Customer

customer = Customer.create(GLOBAL_CUSTOMER_ID, GLOBAL_CUSTOMER_ID, "INFRA", service_
↪subscriptions=["service_type"])
```

4.1.9 Connect service subscription to cloud region and tenant

```
# We assume here that the service subscription has been already created
# and cloud region has a tenant
from onapsdk.aai.business import Customer
from onapsdk.aai.cloud_infrastructure import CloudRegion, Tenant

customer = Customer.get_by_global_customer_id(GLOBAL_CUSTOMER_ID)
service_subscription = next(customer.service_subscriptions)
cloud_region = CloudRegion.get_by_id(
    cloud_owner=CLOUD_OWNER,
    cloud_region_id=CLOUD_REGION
)
tenant = next(cloud_region.tenants)
service_subscription.link_to_cloud_region_and_tenant(cloud_region, tenant)
```

4.1.10 Add Cloud SIte entry to SO Catalog DB

```
from onapsdk.so.so_db_adapter import IdentityService, SoDbAdapter

identity_service = IdentityService(identity_id="mc_test_identity_1_KEYSTONE",
                                   url="http://test:5000/v3",
                                   mso_id="test_user",
                                   mso_pass="test_password_encrypted",
                                   roject_domain_name="Default",
                                   user_domain_name="Default",
                                   identity_server_type="KEYSTONE_V3")
response = SoDbAdapter.add_cloud_site(cloud_region_id="test_region_1",
                                      complex_id="test_clli_1",
                                      identity_service=identity_service,
                                      orchestrator="NULL")
```

4.1.11 Use A&AI bulk API (experimental)

```
from onapsdk.aai.bulk import AaiBulk, AaiBulkRequest
from onapsdk.aai.cloud_infrastructure.cloud_region import CloudRegion
from onapsdk.utils.jinja import jinja_env


for resp in AaiBulk.single_transaction(
    [
        AaiBulkRequest(
            action="put",
            uri=f"/cloud-infrastructure/cloud-regions/cloud-region/aai_bulk_test_
↪cloud_owner_1/aai_bulk_test_cloud_region_id_1",
            body=jinja_env().get_template("cloud_region_create.json.j2").render(cloud_
↪region=CloudRegion(
                    cloud_owner="aai_bulk_test_cloud_owner_1",
                    cloud_region_id="aai_bulk_test_cloud_region_id_1",
                    orchestration_disabled=False,
                    in_maint=False
                ))
        ),
        AaiBulkRequest(
            action="put",
            uri=f"/cloud-infrastructure/cloud-regions/cloud-region/aai_bulk_test_
↪cloud_owner_2/aai_bulk_test_cloud_region_id_2",
            body=jinja_env().get_template("cloud_region_create.json.j2").render(cloud_
↪region=CloudRegion(
                    cloud_owner="aai_bulk_test_cloud_owner_2",
                    cloud_region_id="aai_bulk_test_cloud_region_id_2",
                    orchestration_disabled=False,
                    in_maint=False
                ))
        )
    ]
):
    print(resp)
```

4.2 Design time

4.2.1 Onboard a Vendor

```
from onapsdk.vendor import Vendor
vendor = Vendor(name="myVendor")
vendor.onboard()
```

4.2.2 Onboard a VSP

You will need the package of the VSP to onboard.

```
from onapsdk.sdc.vendor import Vendor
from onapsdk.sdc.vsp import Vsp

# We assume here that the Vendor has been already onboarded
vendor = Vendor(name="myVendor")
vendor.onboard()
vsp = Vsp(name="myVSP", vendor=vendor, package=open(PATH_TO_PACKAGE, 'rb'))
vsp.onboard()
```

4.2.3 Create new VSP version

You will need the package of the VSP to update.

```
from onapsdk.sdc.vendor import Vendor
from onapsdk.sdc.vsp import Vsp

# We assume here that the Vendor has been already onboarded
vsp = Vsp(name="myVSP")
vsp.create_new_version()
vsp.update_package(open(PATH_TO_PACKAGE, 'rb'))
vsp.onboard()
```

4.2.4 Onboard a VF

```
from onapsdk.sdc.vsp import Vsp
from onapsdk.sdc.vf import Vf

# We assume here that the VSP has been already onboarded
vsp = Vsp(name="myVSP")
vf = Vf(name="myVF", vsp=vsp)
vf.onboard()
```

4.2.5 Onboard a VF with properties assignement

```
from onapsdk.sdc.properties import Property
from onapsdk.sdc.vsp import Vsp
from onapsdk.sdc.vf import Vf

# We assume here that the VSP has been already onboarded
vsp = Vsp(name="myVSP")
property_1 = Property(
   name="prop1",
   property_type="string",
   value="test"
)
property_2 = Property(
   name="prop2",
   property_type="integer"
)
vf = Vf(name="myVF",
        vsp=vsp,
        properties=[
           property_1,
           property_2
        ],
        inputs=[property_1])
vf.onboard()
```

4.2.6 Onboard a VF with Deployment Artifact

```
from onapsdk.sdc.properties import Property
from onapsdk.sdc.vsp import Vsp
from onapsdk.sdc.vf import Vf

logger.info("******** Onboard Vendor *******")
vendor = Vendor(name="my_Vendor")
vendor.onboard()

# We assume here that the VSP has been already onboarded
vsp = Vsp(name="myVSP")

logger.info("******** Onboard VF *******")
vf = Vf(name="myVF")
vf.vsp = vsp
vf.create()

logger.info("******** Upload Artifact *******")
vf.add_deployment_artifact(artifact_type="CONTROLLER_BLUEPRINT_ARCHIVE",
                           artifact_name="CBA.zip",
                           artifact_label="vfwcds",
                           artifact="dir/CBA_enriched.zip")

vf.onboard()
```

4.2.7 Onboard a VF with it's component's property input

```
from onapsdk.sdc.properties import ComponentProperty
from onapsdk.sdc.vsp import Vsp
from onapsdk.sdc.vf import Vf
from onapsdk.sdc.vfc import Vfc

# We assume here that the VSP has been already onboarded
vsp = Vsp(name="myVSP")

vfc = Vfc(name="AllottedResource")

logger.info("******** Onboard VF *******")
vf = Vf(name="myVF")
vf.vsp = vsp
vf.create()
vf.add_resource(vfc)
vfc_comp = vf.get_component(vfc)
comp_prop = vfc_comp.get_property("min_instances")
comp_prop.value = 11
vf.declare_input(comp_prop)

vf.onboard()
```

4.2.8 Onboard a PNF with VSP

```
from onapsdk.sdc.pnf import Pnf
from onapsdk.sdc.vendor import Vendor

logger.info("******** Onboard Vendor *******")
vendor = Vendor(name="my_Vendor")
vendor.onboard()

# We assume here that the VSP has been already onboarded
vsp = Vsp(name="myVSP")

logger.info("******** Onboard PNF *******")
pnf = PNF(name="myPNF")
pnf.vsp = vsp
pnf.onboard()
```

4.2.9 Onboard a PNF with Deployment Artifact (without VSP)

```
from onapsdk.sdc.vendor import Vendor
from onapsdk.sdc.pnf import Pnf

logger.info("******** Onboard Vendor *******")
vendor = Vendor(name="my_Vendor")
vendor.onboard()

logger.info("******** Onboard PNF *******")
pnf = Pnf(name=PNF, vendor=vendor)
pnf.create()
```

(continues on next page)

(continued from previous page)

```
logger.info("******** Upload Artifact *******")
pnf.add_deployment_artifact(artifact_type=ARTIFACT_TYPE,
                            artifact_name=ARTIFACT_NAME,
                            artifact_label=ARTIFACT_LABEL,
                            artifact=ARTIFACT_FILE_PATH)
pnf.onboard()
```

4.2.10 Onboard a Service

```
from onapsdk.sdc.vf import Vf
from onapsdk.sdc.service import Service

# We assume here that the VF has been already onboarded
vf = Vf(name="myVF")
service = Service(name="myService", resources=[vf])
service.onboard()
```

4.2.11 Onboard a Service with properties assignement

```
from onapsdk.sdc.properties import Property
from onapsdk.sdc.vf import Vf
from onapsdk.sdc.service import Service

# We assume here that the VF has been already onboarded
vf = Vf(name="myVF")
property_1 = Property(
   name="prop1",
   property_type="string",
   value="test"
)
property_2 = Property(
   name="prop2",
   property_type="integer",
   declare_input=True
)
service = Service(name="myService",
                  resources=[vf],
                  properties=[
                     property_1,
                     property_2
                  ],
                  inputs=[property_1])
service.onboard()
```

4.2.12 Onboard a Service with Nested inputs

```
from onapsdk.sdc.properties import NestedInput
from onapsdk.sdc.vf import Vf
from onapsdk.sdc.service import Service

# We assume here that the VF has been already onboarded
vf = Vf(name="myVF")
inp = vf.get_input("input_name_we_want_to_declare_in_service")
service = Service(name="myService",
                  resources=[vf],
                  inputs=[NestedInput(vf, inp)])
service.onboard()
```

4.2.13 Onboard a Service with VL

```
from onapsdk.sdc.vl import VL
from onapsdk.sdc.service import Service

# No VF needed, but you need to be sure that Vl with given
# name exists in SDC
vl = Vl(name="Generic NeutronNet")
service = Service(name="myServiceWithVl", resources=[vl])
service.onboard()
```

4.2.14 Onboard a Service with custom category

```
from onapsdk.sdc.category_management import ServiceCategory
from onapsdk.sdc.vf import Vf
from onapsdk.sdc.service import Service

# Let's create a custom category
CATEGORY_NAME = "Python ONAP SDK category"
ServiceCategory.create(name=CATEGORY_NAME)

# We assume here that the VF has been already onboarded
# Create a service with category we created few lines above
vf = Vf(name="myVF")
service = Service(name="myService", resources=[vf], category=CATEGORY_NAME)
service.onboard()
```

4.2.15 Onboard an Artifact for an embedded VF

All SDC artifact types are supported

```
from onapsdk.service import Service

# We assume here that the Service has been already onboarded
# with a Vnf
service = Service(name="myService")
# We load artifact data
data = open("{}/myArtifact.yaml".format(os.path.dirname(os.path.abspath(__file__))),
↪'rb').read()
```

(continues on next page)

(continued from previous page)

```
# We add the artifact to the service Vnf
#
svc.add_artifact_to_vf(vnf_name="myVnf",
                       artifact_type="DCAE_INVENTORY_BLUEPRINT",
                       artifact_name="myArtifact.yaml",
                       artifact=data)
```

4.2.16 Onboard a Service with Deployment Artifact

```
from onapsdk.sdc.service import Service

svc = Service(name="myService")

logger.info("******** Upload Artifact *******")
svc.add_deployment_artifact(artifact_type="OTHER",
                            artifact_name="eMBB.zip",
                            artifact_label="embbcn",
                            artifact="dir/eMBB.zip")

svc.onboard()
```

4.2.17 Onboard a Service with a CBA blueprint for Macro Instantiation

```
from onapsdk.sdc.service import Service, ServiceInstantiationType

# Set CBA variables and artifact level
# Must match to values in the CBA TOSCA.meta file
SDNC_TEMPLATE_NAME = "vFW-CDS"
SDNC_TEMPLATE_VERSION = "1.0.0"
SDNC_ARTIFACT_NAME = "vnf"

svc = Service(name="myService",
              instantiation_type=ServiceInstantiationType.MACRO)

svc.create()

logger.info("*** add a VF, which includes a CBA blueprint ***")
svc.add_resource(vf)

logger.info("******** Set SDNC properties for VF ********")
component = svc.get_component(vf)
prop = component.get_property("sdnc_model_version")
prop.value = SDNC_TEMPLATE_NAME
prop = component.get_property("sdnc_artifact_name")
prop.value = SDNC_ARTIFACT_NAME
prop = component.get_property("sdnc_model_name")
prop.value = SDNC_TEMPLATE_NAME
prop = component.get_property("controller_actor")
prop.value = "CDS"
prop = component.get_property("skip_post_instantiation_configuration")
prop.value = False
```

(continues on next page)

(continued from previous page)

```
logger.info("******** Onboard Service *******")
svc.checkin()
svc.onboard()
```

4.3 Instantiation

4.3.1 Create business objects

```
from onapsdk.vid import LineOfBusiness, OwningEntity, Platform, Project

vid_owning_entity = OwningEntity.create(OWNING_ENTITY)
vid_project = Project.create(PROJECT)
vid_platform = Platform.create(PLATFORM)
vid_line_of_business = LineOfBusiness.create(LINE_OF_BUSINESS)
```

4.3.2 Instantiate a service (ALaCarte)

```
import time
from onapsdk.aai.cloud_infrastructure import CloudRegion, Tenant
from onapsdk.aai.business import Customer
from onapsdk.service import Service
from onapsdk.vid import LineOfBusiness, OwningEntity, Platform, Project
from onapsdk.so.instantiation import ServiceInstantiation

# We assume that:
#   - service is onboarded,
#   - cloud region, customer, owning_entity and project have been already created,
#   - cloud region has at least one tenant
#   - customer has service subscription
#   - service subscription is connected with cloud region and tenant
SERVICE_INSTANCE_NAME = "vFW-AlaCarte-1"

service = Service(name="myService")
customer = Customer.get_by_global_customer_id(GLOBAL_CUSTOMER_ID)
cloud_region = CloudRegion.get_by_id(
    cloud_owner=CLOUD_OWNER,
    cloud_region_id=CLOUD_REGION
)
tenant = next(cloud_region.tenants)
vid_owning_entity = OwningEntity(OWNING_ENTITY)
owning_entity = AaiOwningEntity.get_by_owning_entity_name(OWNING_ENTITY)
vid_project = Project(PROJECT)

service_instantiation = ServiceInstantiation.instantiate_so_ala_carte(
    service,
    cloud_region,
    tenant,
    customer,
    owning_entity,
    vid_project,
    service_instance_name=SERVICE_INSTANCE_NAME
```

(continues on next page)

(continued from previous page)

```
)
service_instantiation.wait_for_finish():
    print("Success")
else:
    print("Instantiation failed, check logs")
```

4.3.3 Instantiate a service (Macro)

```
import time
from onapsdk.aai.cloud_infrastructure import CloudRegion, Tenant
from onapsdk.aai.business import Customer
from onapsdk.service import Service
from onapsdk.vid import LineOfBusiness, OwningEntity, Platform, Project
from onapsdk.so.instantiation import (
    ServiceInstantiation,
    VnfInstantiation,
    InstantiationParameter,
    VnfParameters,
    VfmoduleParameters
)

...
VSPNAME = "vfwcds_VS"
VFNAME = "vfwcds_VF"
...
vf = Vf(name=VFNAME)
...

# We assume that:
#   - service is onboarded,
#   - cloud region, customer, owning_entity and project have been already created,
#   - cloud region has at least one tenant
#   - customer has service subscription
#   - service subscription is connected with cloud region and tenant
SERVICE_INSTANCE_NAME = "vFW-Macro-1"

service = Service(name="myMacroService")
customer = Customer.get_by_global_customer_id(GLOBAL_CUSTOMER_ID)
cloud_region = CloudRegion.get_by_id(
    cloud_owner=CLOUD_OWNER,
    cloud_region_id=CLOUD_REGION
)
tenant = next(cloud_region.tenants)
vid_owning_entity = OwningEntity(OWNING_ENTITY)
owning_entity = AaiOwningEntity.get_by_owning_entity_name(OWNING_ENTITY)
vid_project = Project(PROJECT)

###########################################################################
######## VFModule parameters ##############################################
###########################################################################
vfm_base=[
   InstantiationParameter(name="sec_group", value=TENANT_SEC_GROUP),
   InstantiationParameter(name="public_net_id", value=PUBLIC_NET)
]
```

(continues on next page)

(continued from previous page)

```
vfm_vsn=[
   InstantiationParameter(name="sec_group", value=TENANT_SEC_GROUP),
   InstantiationParameter(name="public_net_id", value=PUBLIC_NET)
]

vfm_vfw=[
   InstantiationParameter(name="sec_group", value=TENANT_SEC_GROUP),
   InstantiationParameter(name="public_net_id", value=PUBLIC_NET)
]

vfm_vpkg=[
   InstantiationParameter(name="sec_group", value=TENANT_SEC_GROUP),
   InstantiationParameter(name="public_net_id", value=PUBLIC_NET)
]

base_paras=VfmoduleParameters("base_template",vfm_base)
vpkg_paras=VfmoduleParameters("vpkg",vfm_vpkg)
vsn_paras=VfmoduleParameters("vsn",vfm_vsn)
vfw_paras=VfmoduleParameters("vfw",vfm_vfw)

##########################################################################
######## VNF parameters ##################################################
##########################################################################

vnf_vfw=[
   InstantiationParameter(name="onap_private_net_id", value=ONAP_PRIVATE_NET),
   InstantiationParameter(name="onap_private_subnet_id", value=ONAP_PRIVATE_SUBNET),
   InstantiationParameter(name="pub_key", value="ssh-rsa AAAAB3NzaC1yc2EAA\
AADAQABAAABAQDFBOB1Ea2yej68aqIQw10kEsVf+rNoxT39qrV8JvvTK2yhkniQka1t2oD9h6DlXOL\
M3HJ6nBegWjOasJmIbminKZ6wvmxZrDVFJXp9Sn1gni0vtEnlDgH14shRUrFDYO0PYjXRHoj7QXZMY\
xtAdFSbzGuCsaTLcV/xchLBQmqZ4AGhMIiYMfJJF+Ygy0lbgcVmT+8DH7kUUt8SAdh2rRsYFwpKANn\
QJyPV1dBNuTcD0OW1hEOhXnwqH28tjfb7uHJzTyGZlTmwTs544teTNz5B9L4yT3XiCAlMcaLOBMfBT\
KRIse+NkiTb+tc60JNnEYR6MqZoqTea/w+YBQaIMcil"),
   InstantiationParameter(name="image_name", value=IMAGE_NAME),
   InstantiationParameter(name="flavor_name", value=FLAVOR_NAME),
   InstantiationParameter(name="sec_group", value=TENANT_SEC_GROUP),
   InstantiationParameter(name="install_script_version", value="1.4.0-SNAPSHOT"),
   InstantiationParameter(name="demo_artifacts_version", value="1.4.0-SNAPSHOT"),
   InstantiationParameter(name="cloud_env", value=CLOUD_TYPE),
   InstantiationParameter(name="public_net_id", value=PUBLIC_NET),
   InstantiationParameter(name="aic-cloud-region", value=CLOUD_REGION)
]

vnf_paras=VnfParameters("vfwcds_VF", vnf_vfw,
          [base_paras, vpkg_paras, vsn_paras, vfw_paras])

# You must define for each VNF and its vFModule the parameters,
# otherwise they stay empty.
# The matching critera are:
# - VnfParameters.name must match VNF ModelInstanceName
#   (see above "vfwcds_VF")
# - VfmoduleParameters.name must match substring in vfModule "instanceName"
#   (e.g. "vfwcds_vf0..VfwcdsVf..vsn..module-1")

service_instantiation = ServiceInstantiation.instantiate_macro(
    service,
    cloud_region,
```

(continues on next page)

(continued from previous page)

```
    tenant,
    customer,
    owning_entity,
    vid_project,
    vid_line_of_business,
    vid_platform,
    service_instance_name=SERVICE_INSTANCE_NAME,
    vnf_parameters=[vnf_paras]
)

service_instantiation.wait_for_finish():
    print("Success")
else:
    print("Instantiation failed, check logs")
```

4.3.4 Instantiate a service using SO service template (Macro)

To provide more control on the SO macro instantiation, you can define your service as follows:

```
myservice:
    subscription_service_type: myservice
    vnfs:
        - model_name: myvfmodel
          instance_name: myfirstvnf
          parameters:
              param1: value1
          processing_priority: 1
          vf_modules:
              - instance_name: mysecondvfm
                model_name: base
                processing_priority: 2
                parameters:
                    param-vfm1: value-vfm1
              - instance_name: myfirstvfm
                model_name: base
                processing_priority: 1
                parameters:
                    param-vfm1: value-vfm1
        - model_name: myvfmodel
          instance_name: mysecondvnf
          parameters:
              param1: value1
          processing_priority: 2
          vf_modules:
              - instance_name: myfirstvfm
                model_name: base
                processing_priority: 1
                parameters:
                    param-vfm1: value-vfm1
              - instance_name: mysecondvfm
                model_name: base
                processing_priority: 2
                parameters:
                    param-vfm1: value-vfm1
```

```
from onapsdk.aai.business import Customer, OwningEntity, Project, LineOfBusiness,␣
↪Platform
from onapsdk.aai.cloud_infrastructure import CloudRegion
from onapsdk.sdc.service import Service
from onapsdk.so.instantiation import ServiceInstantiation
from yaml import load

so_yaml_service = "/path/to/yaml/service"
with open(so_yaml_service, "r") as yaml_template:
    so_service_data = load(yaml_template)

# We assume that:
#   - service is onboarded,
#   - cloud region, customer, owning_entity and project have been already created,
#   - cloud region has at least one tenant
#   - customer has service subscription
#   - service subscription is connected with cloud region and tenant

service = Service(next(iter(so_service_data.keys())))
so_service = SoService.load(so_service_data[service.name])
SERVICE_INSTANCE_NAME = "my_svc_instance_name"

customer = Customer.get_by_global_customer_id(GLOBAL_CUSTOMER_ID)
cloud_region = CloudRegion.get_by_id(
    cloud_owner=CLOUD_OWNER,
    cloud_region_id=CLOUD_REGION
)
tenant = next(cloud_region.tenants)
owning_entity = OwningEntity(OWNING_ENTITY)
project = Project(PROJECT)
line_of_business = LineOfBusiness(LINE_OF_BUSINESS)
platform = Platform(PLATFORM)

service_instantiation = ServiceInstantiation.instantiate_macro(
    sdc_service=service,
    customer=customer,
    owning_entity=owning_entity,
    project=project,
    line_of_business=line_of_business,
    platform=platform,
    cloud_region=cloud_region,
    tenant=tenant,
    service_instance_name=SERVICE_INSTANCE_NAME,
    so_service=so_service
)
```

4.3.5 Instantiate VNF (Macro)

Since ONAP Istanbul the creation or deletion of VNFs in macro mode is supported. Examples below:

```
import time
from onapsdk.aai.business import Customer
from onapsdk.vid import LineOfBusiness, Platform

# We assume that
#   - service has been already instantiated,
```

(continues on next page)

(continued from previous page)

```
#   - line of business and platform are created

SERVICE_INSTANCE_NAME = "service_instance_demo"
VNF_INSTANCE_NAME = "new_vnf_instance"

customer = Customer.get_by_global_customer_id(GLOBAL_CUSTOMER_ID)
service_subscription = next(customer.service_subscriptions)
service_instance = service_subscription.get_service_instance_by_name(SERVICE_INSTANCE_
↪NAME)
vnf = service_subscription.sdc_service.vnfs[0]
vid_line_of_business = LineOfBusiness.create(LINE_OF_BUSINESS)
vid_platform = Platform.create(PLATFORM)

###########################################################################
######## VFModule parameters ##############################################
###########################################################################

myfirstvfm_params = [
    InstantiationParameter(name="param-vfm1", value="value-vfm1")
]

vf1_params = VfmoduleParameters("myfirstvfm", myfirstvfm_params)

###########################################################################
######## VNF parameters ###################################################
###########################################################################

vnf_param_list = [
    InstantiationParameter(name="param1", value="value1")
]

vnf_paras = VnfParameters("myvfmodel", vnf_param_list, [vf1_params])

vnf_instantiation = service_instance.add_vnf(
    vnf=vnf,
    line_of_business=vid_line_of_business,
    platform=vid_platform,
    vnf_instance_name=VNF_INSTANCE_NAME,
    vnf_parameters=[vnf_paras],
    a_la_carte=False
)

vnf_instantiation.wait_for_finish():
    print("Success")
else:
    print("Instantiation failed, check logs")
```

4.3.6 Instantiate VNF using SO service template (Macro)

To provide more control on the SO macro instantiation for new vnf, you can define your vnf as follows:

```
model_name: myvfmodel
instance_name: mynewvnf
parameters:
    param1: value1
vf_modules:
    - instance_name: mysecondvfm
      model_name: base
      processing_priority: 2
      parameters:
          param-vfm1: value-vfm1
    - instance_name: myfirstvfm
      model_name: base
      processing_priority: 1
      parameters:
          param-vfm1: value-vfm1
```

```
import time
from onapsdk.aai.business import Customer
from onapsdk.vid import LineOfBusiness, Platform

SERVICE_INSTANCE_NAME = "service_instance_demo"
VNF_INSTANCE_NAME = "new_vnf_instance"

# We assume that
#   - service has been already instantiated,
#   - line of business and platform are created

customer = Customer.get_by_global_customer_id(GLOBAL_CUSTOMER_ID)
service_subscription = next(customer.service_subscriptions)
service_instance = service_subscription.get_service_instance_by_name(SERVICE_INSTANCE_
↪NAME)
vnf = service_subscription.sdc_service.vnfs[0]
vid_line_of_business = LineOfBusiness.create(LINE_OF_BUSINESS)
vid_platform = Platform.create(PLATFORM)

so_yaml_vnf = "/path/to/yaml/vnf"
with open(so_yaml_vnf, "r") as yaml_template:
    so_vnf_data = load(yaml_template)

so_vnf = SoServiceVnf.load(so_vnf_data)

vnf_instantiation = service_instance.add_vnf(
    vnf=vnf,
    line_of_business=vid_line_of_business,
    platform=vid_platform,
    vnf_instance_name=VNF_INSTANCE_NAME,
    so_vnfs=so_vnfs,
    a_la_carte=False
)

vnf_instantiation.wait_for_finish():
    print("Success")
else:
    print("Instantiation failed, check logs")
```

4.3.7 Instantiate VNF (ALaCarte)

```
import time
from onapsdk.aai.business import Customer
from onapsdk.vid import LineOfBusiness, Platform

# We assume that
#   - service has been already instantiated,
#   - line of business and platform are created

customer = Customer.get_by_global_customer_id(GLOBAL_CUSTOMER_ID)
service_subscription = next(customer.service_subscriptions)
service_instance = service_subscription.get_service_instance_by_name(SERVICE_INSTANCE_
↪NAME)
vnf = service_subscription.sdc_service.vnfs[0]
vid_line_of_business = LineOfBusiness.create(LINE_OF_BUSINESS)
vid_platform = Platform.create(PLATFORM)
vnf_instantiation = service_instance.add_vnf(vnf, vid_line_of_business, vid_platform)
vnf_instantiation.wait_for_finish():
    print("Success")
else:
    print("Instantiation failed, check logs")
```

4.3.8 Instantiate Vf module (ALaCarte)

```
import time
from onapsdk.aai.business import Customer

customer = Customer.get_by_global_customer_id(GLOBAL_CUSTOMER_ID)
service_subscription = next(customer.service_subscriptions)
service_instance = service_subscription.get_service_instance_by_name(SERVICE_INSTANCE_
↪NAME)
vnf_instance = next(service_instance.vnf_instances)
vf_module = vnf_instance.vnf.vf_module
vf_module_instantiation = vnf_instance.add_vf_module(
    vf_module,
    vnf_parameters=[
        VnfParameter(name="parameter1", value="parameter1_value"),
        VnfParameter(name="parameter2", value="parameter2_value
    ]
)
vf_module_instantiation.wait_for_finish():
    print("Success")
else:
    print("Instantiation failed, check logs")
```

4.3.9 Instantiate VI module (ALaCarte)

```
import time
from onapsdk.aai.business import Customer
from onapsdk.vid import LineOfBusiness, Platform

# We assume that
#   - service has been already instantiated,
#   - line of business and platform are created

customer = Customer.get_by_global_customer_id(GLOBAL_CUSTOMER_ID)
service_subscription = next(customer.service_subscriptions)
service_instance = service_subscription.get_service_instance_by_name(SERVICE_INSTANCE_
↪NAME)

logger.info("******** Get 1st Network in Service Model *******")
network = service_subscription.sdc_service.networks[0]

logger.info("******** Create Network *******")
sn=Subnet(name="my_subnet",
          start_address="10.0.0.1",
          cidr_mask="24",
          gateway_address="10.0.0.1)

vid_line_of_business = LineOfBusiness.create(LINE_OF_BUSINESS)
vid_platform = Platform.create(PLATFORM)

network_instantiation = service_instance.add_network(network, vid_line_of_business,
                        vid_platform, network_instance_name="my_net", subnets=[sn])

if network_instantiation.wait_for_finish():
    print("Success")
else:
    print("Instantiation failed, check logs")
```

4.4 Instantiated resources deletion

4.4.1 Service, vnf and vf module deletion

```
from onapsdk.aai.business import Customer

customer = Customer.get_by_global_customer_id(GLOBAL_CUSTOMER_ID)
service_subscription = next(customer.service_subscriptions)
service_instance = service_subscription.get_service_instance_by_name(SERVICE_INSTANCE_
↪NAME)
for vnf_instance in service_instance.vnf_instances:
    for vf_module_instance in vnf_instance.vf_modules:
        vf_module_deletion_request = vf_module_instance.delete()
        while not vf_module_deletion.finished:
            time.sleep(10)

    vnf_instance_deletion_request = vnf_instance.delete()
    while not vnf_instance_deletion_request.finished:
        time.sleep(10)
```

(continues on next page)

(continued from previous page)

```
service_instance_deletion_request = service_instance.delete()
if service_instance_deletion_request.wait_for_finish():
    print("Service instance deleted")
else:
    print("Service deletion failed, check logs"
```

4.5 CDS

4.5.1 Preparation for CDS tests

To enable CDS Enrichment in an ONAP Frankfurt environment the NodePort 30449 for the CDS Blueprint Processor API service needs to be opened

1. Check existing CDS Services:

```
ubuntu@control01:~$ kubectl get service -n onap|grep cds-blueprints-processor-http
cds-blueprints-processor-http      ClusterIP  10.43.101.198   <none>  8080/TCP
```

2. Change NodePort to CDS cds-blueprints-processor-http

   Add the "nodePort" under "ports" section and change "type" from "ClusterIP" to "NodePort"

```
ubuntu@control01:~$ kubectl edit service cds-blueprints-processor-http -n onap

apiVersion: v1
kind: Service
metadata:
  creationTimestamp: "2020-07-23T02:57:36Z"
  labels:
    app: cds-blueprints-processor
    chart: cds-blueprints-processor-6.0.0
    heritage: Tiller
    release: onap
  name: cds-blueprints-processor-http
  namespace: onap
  resourceVersion: "10256"
  selfLink: /api/v1/namespaces/onap/services/cds-blueprints-processor-http
  uid: 6f065c03-4563-4d64-b6f5-a8892226c909
spec:
  clusterIP: 10.43.101.198
  ports:
  - name: blueprints-processor-http
    nodePort: 30449       -> add line
    port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    app: cds-blueprints-processor
    release: onap
  sessionAffinity: None
  type: ClusterIP -> change to NodePort
status:
  loadBalancer: {}
```

3. Verify NodePort to CDS cds-blueprints-processor-http

```
ubuntu@control01:~$ kubectl get service -n onap|grep cds-blueprints-processor-http
cds-blueprints-processor-http      NodePort    10.43.101.198   <none> 8080:30449/
↪TCP
```

4. Load ModelType via Bootstrap

```
curl --location --request POST 'http://<k8s-host>:30449/api/v1/blueprint-model/
↪bootstrap' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic Y2NzZGthcHBzOmNjc2RrYXBwcw==' \
--data-raw '{
"loadModelType" : true,
"loadResourceDictionary" : false,
"loadCBA" : false
}'
```

4.5.2 Load blueprint from file

```
from onapsdk.cds import Blueprint
blueprint = Blueprint.load_from_file("<< path to CBA file >>")
```

4.5.3 Enrich blueprint and save

```
enriched_blueprint = blueprint.enrich()
enriched_blueprint.save("<< path to dest file >>")
```

4.5.4 Publish blueprint

```
enriched_blueprint.publish()
```

4.5.5 Generate data dictionary from blueprint

The method to generate data dictionaries based on the blueprint mappings. As the result it returns a data dictionaries set with valid structure, but some additional actions may be needed. Data dictionary input has to be filled by the user if the type is neither "source-input" nor "source-default". Things, which are needed to be filled are marked by << *FILL* >> mark. If the blueprint you are using has only "source-input" or "source-default" input types, the generated data dictionary set is ready to upload to CDS.

```
generated_dd: DataDictionarySet = blueprint.get_data_dictionaries()
generated_dd.save_to_file("<< path to dest file >>")
```

4.5.6 Load data dictionary set from file

```
from onapsdk.cds import DataDictionarySet
dd_set = DataDictionarySet.load_from_file("<< path to dd file >>")
```

4.5.7 Upload data dictionary set

```
dd_set.upload()
```

4.5.8 Retrieve Blueprint Models from CDS

1. All

```
from onapsdk.cds import BlueprintModel
all_blueprint_models = BlueprintModel.get_all()
```

1. Selected by id of Blueprint Model

```
blueprint_model = BlueprintModel.get_by_id(blueprint_model_id='11111111-1111-1111-
↪1111-111111111111')
```

1. Selected by name and version of Blueprint Model

```
blueprint_model = BlueprintModel.get_by_name_and_version(blueprint_name='test_name',␣
↪blueprint_version='1.0.0')
```

4.5.9 Delete Blueprint Model

```
blueprint_model.delete()
```

4.5.10 Download Blueprint Model

```
blueprint_model.save(dst_file_path='/tmp/blueprint.zip')
```

4.5.11 Get Blueprint object for Blueprint Model

After that, all operation for blueprint object, like execute blueprint workflow etc. can be executed.

```
blueprint = blueprint_model.get_blueprint()
```

4.6 CPS

4.6.1 Create dataspace

```
from onapsdk.cps import Dataspace
dataspace: Dataspace = Dataspace.create(dataspace_name="test_dataspace")
```

4.6.2 Create schema set

```
from onapsdk.cps import Dataspace, SchemaSet
dataspace: Dataspace = Dataspace(name="test_dataspace")
with Path("schema_set_zip_file.zip").open("rb") as zip_file:
    schema_set: SchemaSet = dataspace.create_schema_set("test_schemaset", zip_file)
```

4.6.3 Create anchor

```
from onapsdk.cps import Anchor, Dataspace, SchemaSet
dataspace: Dataspace = Dataspace(name="test_dataspace")
schema_set: SchemaSet = dataspace.get_schema_set("test_schemaset")
anchor: Anchor = dataspace.create_anchor(schema_set, "test_anchor")
```

CHAPTER

FIVE

REAL LIFE SCRIPT EXAMPLES

5.1 E2E Instantiation of vFW (a'la carte)

```
import logging
import time
from uuid import uuid4
from onapsdk.aai.aai_element import AaiElement
from onapsdk.aai.cloud_infrastructure import (
    CloudRegion,
    Complex,
    Tenant
)
from onapsdk.aai.service_design_and_creation import (
    Service as AaiService
)
from onapsdk.aai.business import (
    ServiceInstance,
    VnfInstance,
    VfModuleInstance,
    ServiceSubscription,
    Customer,
    OwningEntity as AaiOwningEntity
)
from onapsdk.so.instantiation import (
    ServiceInstantiation,
    VnfInstantiation,
    VnfParameter
)
from onapsdk.sdc import SDC
from onapsdk.sdc.vendor import Vendor
from onapsdk.sdc.vsp import Vsp
from onapsdk.sdc.vf import Vf
from onapsdk.sdc.service import Service
import onapsdk.constants as const
import os
from onapsdk.vid import LineOfBusiness, OwningEntity, Platform, Project

logger = logging.getLogger("")
logger.setLevel(logging.INFO)
fh = logging.StreamHandler()
fh_formatter = logging.Formatter('%(asctime)s %(levelname)s %(lineno)d:%(filename)s(
↪%(process)d) - %(message)s')
fh.setFormatter(fh_formatter)
```

(continues on next page)

(continued from previous page)

```
logger.addHandler(fh)



# Create required A&AI resources
VENDOR = "VNFVendor"
VSPFILE = "vsp/vfw.zip"
VSPNAME = "vfw_VSP"
VFNAME = "vfw_VF"
SERVICENAME = "vfw_SERVICE"

GLOBAL_CUSTOMER_ID = ""  # FILL ME
COMPLEX_PHYSICAL_LOCATION_ID = ""  # FILL ME
COMPLEX_DATA_CENTER_CODE = ""  # FILL ME

CLOUD_OWNER = ""  # FILL ME
CLOUD_REGION = ""  # FILL ME

VIM_USERNAME = ""  # FILL ME
VIM_PASSWORD = ""  # FILL ME
VIM_SERVICE_URL = ""  # FILL ME

TENANT_NAME = ""  # FILL ME
OWNING_ENTITY = ""  # FILL ME
PROJECT = ""  # FILL ME
PLATFORM = ""  # FILL ME
LINE_OF_BUSINESS = ""  # FILL ME

SERVICE_INSTANCE_NAME = "vFW-Instance"
SERVICE_DELETION = True

logger.info("******************************")
logger.info("******** SERVICE DESIGN *******")
logger.info("******************************")

logger.info("******** Onboard Vendor *******")
vendor = Vendor(name=VENDOR)
vendor.onboard()

logger.info("******** Onboard VSP *******")
mypath = os.path.dirname(os.path.realpath(__file__))
myvspfile = os.path.join(mypath, VSPFILE)
vsp = Vsp(name=VSPNAME, vendor=vendor, package=open(myvspfile, 'rb'))
vsp.onboard()

logger.info("******** Onboard VF *******")
vf = Vf(name=VFNAME)
vf.vsp = vsp
vf.onboard()

logger.info("******** Onboard Service *******")
svc = Service(name=SERVICENAME, resources=[vf])
svc.onboard()

logger.info("******** Check Service Distribution *******")
distribution_completed = False
nb_try = 0
```

(continues on next page)

(continued from previous page)

```
nb_try_max = 10
while distribution_completed is False and nb_try < nb_try_max:
    distribution_completed = svc.distributed
    if distribution_completed is True:
       logger.info("Service Distribution for %s is sucessfully finished",svc.name)
       break
    logger.info("Service Distribution for %s ongoing, Wait for 60 s",svc.name)
    time.sleep(60)
    nb_try += 1

if distribution_completed is False:
    logger.error("Service Distribution for %s failed !!",svc.name)
    exit(1)

logger.info("*******************************")
logger.info("***** RUNTIME PREPARATION *****")
logger.info("*******************************")

logger.info("******** Create Complex *******")
cmplx = Complex.create(
    physical_location_id=COMPLEX_PHYSICAL_LOCATION_ID,
    data_center_code=COMPLEX_DATA_CENTER_CODE,
    name=COMPLEX_PHYSICAL_LOCATION_ID
)

logger.info("******** Create CloudRegion *******")
cloud_region = CloudRegion.create(
    cloud_owner=CLOUD_OWNER,
    cloud_region_id=CLOUD_REGION,
    orchestration_disabled=False,
    in_maint=False,
    cloud_type="openstack",
    cloud_region_version="titanium_cloud",
    cloud_zone="z1",
    complex_name=COMPLEX_PHYSICAL_LOCATION_ID
)

logger.info("******** Link Complex to CloudRegion *******")
cloud_region.link_to_complex(cmplx)

logger.info("******** Add ESR Info to CloudRegion *******")
cloud_region.add_esr_system_info(
    esr_system_info_id=str(uuid4()),
    user_name=VIM_USERNAME,
    password=VIM_PASSWORD,
    system_type="VIM",
    service_url=VIM_SERVICE_URL,
    cloud_domain="Default",
    ssl_insecure=False,
    system_status="active",
    default_tenant=TENANT_NAME
)

logger.info("******** Register CloudRegion to MultiCloud *******")
cloud_region.register_to_multicloud()

logger.info("******** Check MultiCloud Registration *******")
```

(continues on next page)

(continued from previous page)

```
time.sleep(60)
registration_completed = False
nb_try = 0
nb_try_max = 10
while registration_completed is False and nb_try < nb_try_max:
    for tenant in cloud_region.tenants:
        logger.debug("Tenant %s found in %s_%s",tenant.name,cloud_region.cloud_owner,
↪cloud_region.cloud_region_id)
        registration_completed = True
    if registration_completed is False:
        time.sleep(60)
    nb_try += 1

if registration_completed is False:
    logger.error("Registration of Cloud %s_%s failed !!",cloud_region.cloud_owner,
↪cloud_region.cloud_region_id)
    exit(1)
else:
    logger.info("Registration of Cloud %s_%s successful !!",cloud_region.cloud_owner,
↪cloud_region.cloud_region_id)

logger.info("*******************************")
logger.info("**** SERVICE INSTANTIATION ****")
logger.info("*******************************")

logger.info("******** Create Customer *******")
customer = None
for found_customer in list(Customer.get_all()):
    logger.debug("Customer %s found", found_customer.subscriber_name)
    if found_customer.subscriber_name == GLOBAL_CUSTOMER_ID:
        logger.info("Customer %s found", found_customer.subscriber_name)
        customer = found_customer
        break
if not customer:
    customer = Customer.create(GLOBAL_CUSTOMER_ID,GLOBAL_CUSTOMER_ID, "INFRA")

logger.info("******** Find Service in SDC *******")
service = None
services = Service.get_all()
for found_service in services:
    logger.debug("Service %s is found, distribution %s",found_service.name, found_
↪service.distribution_status)
    if found_service.name == SERVICENAME:
        logger.info("Found Service %s in SDC",found_service.name)
        service = found_service
        break

if not service:
    logger.error("Service %s not found in SDC",SERVICENAME)
    exit(1)

logger.info("******** Check Service Subscription *******")
service_subscription = None
for service_sub in customer.service_subscriptions:
    logger.debug("Service subscription %s is found",service_sub.service_type)
    if service_sub.service_type == SERVICENAME:
        logger.info("Service %s subscribed",SERVICENAME)
```

(continues on next page)

(continued from previous page)

```
        service_subscription = service_sub
        break

if not service_subscription:
    logger.info("******** Subscribe Service *******")
    customer.subscribe_service(SERVICENAME)

logger.info("******** Get Tenant *******")
cloud_region = CloudRegion(cloud_owner=CLOUD_OWNER, cloud_region_id=CLOUD_REGION,
                           orchestration_disabled=True, in_maint=False)
tenant = None
for found_tenant in cloud_region.tenants:
    logger.debug("Tenant %s found in %s_%s",found_tenant.name,cloud_region.cloud_
↪owner,cloud_region.cloud_region_id)
    if found_tenant.name == TENANT_NAME:
        logger.info("Found my Tenant %s",found_tenant.name)
        tenant = found_tenant
        break

if not tenant:
    logger.error("tenant %s not found",TENANT_NAME)
    exit(1)

logger.info("******** Connect Service to Tenant *******")
service_subscription = None
for service_sub in customer.service_subscriptions:
    logger.debug("Service subscription %s is found",service_sub.service_type)
    if service_sub.service_type == SERVICENAME:
        logger.info("Service %s subscribed",SERVICENAME)
        service_subscription = service_sub
        break

if not service_subscription:
    logger.error("Service subscription %s is not found",SERVICENAME)
    exit(1)

service_subscription.link_to_cloud_region_and_tenant(cloud_region, tenant)

logger.info("******** Add Business Objects (OE, P, Pl, LoB) in VID *******")
vid_owning_entity = OwningEntity.create(OWNING_ENTITY)
vid_project = Project.create(PROJECT)
vid_platform = Platform.create(PLATFORM)
vid_line_of_business = LineOfBusiness.create(LINE_OF_BUSINESS)

logger.info("******** Add Owning Entity in AAI *******")
owning_entity = None
for oe in AaiOwningEntity.get_all():
    if oe.name == vid_owning_entity.name:
        owning_entity = oe
        break
if not owning_entity:
    logger.info("******** Owning Entity not existing: create *******")
    owning_entity = AaiOwningEntity.create(vid_owning_entity.name, str(uuid4()))

logger.info("******** Instantiate Service *******")
service_instance = None
service_instantiation = None
```

(continues on next page)

(continued from previous page)

```
for se in service_subscription.service_instances:
   if se.instance_name == SERVICE_INSTANCE_NAME:
       service_instance = se
       break
if not service_instance:
    logger.info("******** Service Instance not existing: Instantiate *******")
    # Instantiate service
    service_instantiation = ServiceInstantiation.instantiate_so_ala_carte(
        service,
        cloud_region,
        tenant,
        customer,
        owning_entity,
        vid_project,
        service_instance_name=SERVICE_INSTANCE_NAME
    )
    time.sleep(60)
else:
    logger.info("******** Service Instance already existing *******")

service_instance = None
for se in service_subscription.service_instances:
   if se.instance_name == SERVICE_INSTANCE_NAME:
       service_instance = se
       break
if not service_instance:
    logger.error("******** Service %s instantiation failed",SERVICE_INSTANCE_NAME)
    exit(1)

nb_try = 0
nb_try_max = 10
service_active = False
while service_active is False and nb_try < nb_try_max:
    if service_instance.orchestration_status == "Active":
       logger.info("******** Service Instance %s is active *******",service_instance.
↪name)
       service_active = True
       break
    logger.info("Service %s instantiation not complete,Status:%s, wait 10s",service_
↪instance.name,service_instance.orchestration_status)
    time.sleep(10)
    nb_try += 1

if service_active is False:
    logger.error("Service %s instantiation failed",service_instance.name)
    exit(1)


logger.info("******** Get VNFs in Service Model *******")
vnfs = service_instance.service_subscription.sdc_service.vnfs

logger.info("******** Create VNFs *******")
for vnf in vnfs:
    logger.debug("Check if VNF instance of class %s exist", vnf.name)
    vnf_found = False
    for vnf_instance in service_instance.vnf_instances:
        logger.debug("VNF instance %s found in Service Instance ",vnf_instance.name)
```

(continues on next page)

(continued from previous page)

```
        vnf_found = True
    if vnf_found is False:
        vnf_instantiation = service_instance.add_vnf(vnf, vid_line_of_business, vid_
↪platform)
        while not vnf_instantiation.finished:
            print("Wait for VNF %s instantiation",vnf.name)
            time.sleep(10)


for vnf_instance in service_instance.vnf_instances:
    logger.debug("VNF instance %s found in Service Instance ",vnf_instance.name)
    logger.info("******** Get VfModules in VNF Model *******")
    logger.info("******** Check VF Modules *******")
    vf_module = vnf_instance.vnf.vf_module

    logger.info("******** Create VF Module %s *******",vf_module.name)

    vf_module_instantiation = vnf_instance.add_vf_module(
                               vf_module,
                               vnf_parameters=[
                                 VnfParameter(name="vfw_image_name", value="Ubuntu_
↪1404"),
                                 VnfParameter(name="vpg_image_name", value="Ubuntu_
↪1404"),
                                 VnfParameter(name="vsn_image_name", value="Ubuntu_
↪1404"),
                                 VnfParameter(name="vfw_flavor_name", value="m1.small
↪"),
                                 VnfParameter(name="vpg_flavor_name", value="m1.small
↪"),
                                 VnfParameter(name="vsn_flavor_name", value="m1.small
↪"),
                                 VnfParameter(name="public_net_id", value="admin"),
                                 VnfParameter(name="onap_private_net_id", value="admin
↪"),
                                 VnfParameter(name="onap_private_subnet_id", value=
↪"admin-subnet"),
                                 VnfParameter(name="onap_private_net_cidr", value="10.
↪41.1.0/24"),
                                 VnfParameter(name="vfw_onap_private_ip_0", value="10.
↪41.1.10"),
                                 VnfParameter(name="vpg_onap_private_ip_0", value="10.
↪41.1.11"),
                                 VnfParameter(name="vsn_onap_private_ip_0", value="10.
↪41.1.12"),
                                 VnfParameter(name="sec_group", value="ci-onap-master-
↪vnfs-onap")
                                 ]
                              )
    nb_try = 0
    nb_try_max = 30
    while not vf_module_instantiation.finished and nb_try < nb_try_max:
        logger.info("Wait for vf module instantiation")
        nb_try += 1
        time.sleep(10)
    if vf_module_instantiation.finished:
        logger.info("VfModule %s instantiated",vf_module.name)
```

(continues on next page)

(continued from previous page)

```
    else:
        logger.error("VfModule instantiation %s failed",vf_module.name)

if SERVICE_DELETION is False:
    logger.info("****************************************")
    logger.info("**** No Deletion requested, finished ****")
    logger.info("****************************************")
    exit(0)

logger.info("******************************")
logger.info("**** SERVICE DELETION *********")
logger.info("******************************")
time.sleep(30)

for vnf_instance in service_instance.vnf_instances:
    logger.debug("VNF instance %s found in Service Instance ",vnf_instance.name)
    logger.info("******** Get VF Modules *******")
    for vf_module in vnf_instance.vf_modules:
        logger.info("******** Delete VF Module %s *******",vf_module.name)
        vf_module_deletion = vf_module.delete()

        nb_try = 0
        nb_try_max = 30
        while not vf_module_deletion.finished and nb_try < nb_try_max:
            logger.info("Wait for vf module deletion")
            nb_try += 1
            time.sleep(10)
        if vf_module_deletion.finished:
            logger.info("VfModule %s deleted",vf_module.name)
        else:
            logger.error("VfModule deletion %s failed",vf_module.name)
            exit(1)

    logger.info("******** Delete VNF %s *******",vnf_instance.name)
    vnf_deletion = vnf_instance.delete()

    nb_try = 0
    nb_try_max = 30
    while not vnf_deletion.finished and nb_try < nb_try_max:
        logger.info("Wait for vnf deletion")
        nb_try += 1
        time.sleep(10)
    if vnf_deletion.finished:
        logger.info("VNF %s deleted",vnf_instance.name)
    else:
        logger.error("VNF deletion %s failed",vnf_instance.name)
        exit(1)

logger.info("******** Delete Service %s *******",service_instance.name)
service_deletion = service_instance.delete()

nb_try = 0
nb_try_max = 30
while not service_deletion.finished and nb_try < nb_try_max:
    logger.info("Wait for Service deletion")
    nb_try += 1
    time.sleep(10)
```

(continues on next page)

(continued from previous page)

```
if service_deletion.finished:
    logger.info("Service %s deleted",service_instance.name)
else:
    logger.error("Service deletion %s failed",service_instance.name)
    exit(1)
```

5.2 E2E Instantiation of a Closed Loop

```
#Service already created in this case

logger = logging.getLogger("")
logger.setLevel(logging.INFO)
fh = logging.StreamHandler()
fh_formatter = logging.Formatter('%(asctime)s %(levelname)s %(lineno)d:%(filename)s(
↪%(process)d) - %(message)s')
fh.setFormatter(fh_formatter)
logger.addHandler(fh)


#Constants
SERVICE_NAME = "Test_SDK"
POLICY_NAME = ["MinMax", "FrequencyLimiter"]
LOOP_INSTANCE_NAME = "instance01"
CERT = (PEM, KEY) # you must add clamp cert for AUTHENTIFICATION

Clamp.set_proxy({ 'http': 'socks5h://127.0.0.1:8080', 'https': 'socks5h://127.0.0.
↪1:8080'})
Service.set_proxy({ 'http': 'socks5h://127.0.0.1:8080', 'https': 'socks5h://127.0.0.
↪1:8080'})

logger.info("******************************")
logger.info("******** SERVICE FETCH *******")
logger.info("******************************")

svc = Service(name=SERVICE_NAME)

logger.info("**************************************")
logger.info("******** CLAMP AUTHENTIFICATION *******")
logger.info("**************************************")

Clamp(cert=CERT)

logger.info("**********************************")
logger.info("******** LOOP TEMPLATES CHECK *******")
logger.info("**********************************")

loop_template = Clamp.check_loop_template(service=svc)
if not loop_template:
    logger.error("Loop template for the service %s not found", svc.name)
    exit(1)

logger.info("******************************")
logger.info("******** POLICIES CHECK *******")
logger.info("******************************")
```

(continues on next page)

(continued from previous page)

```
minmax_exists = Clamp.check_policies(policy_name=POLICY_NAME[0],
                                     req_policies=30)
frequency_exists = Clamp.check_policies(policy_name=POLICY_NAME[1],
                                     req_policies=30)
policy_exists = (minmax_exists and frequency_exists)
if not policy_exists:
    logger.error("Couldn't load the policy %s", POLICY_NAME)
    exit(1)

logger.info("**********************************")
logger.info("******** LOOP INSTANTIATION *******")
logger.info("**********************************")

loop = LoopInstance(template=loop_template, name=LOOP_INSTANCE_NAME, details={},␣
↪cert=CERT)
loop.create()
if loop.details:
    logger.info("Loop instance %s successfully created !!", LOOP_INSTANCE_NAME)
else:
    logger.error("An error occured while creating the loop instance")

logger.info("******** UPDATE MICROSERVICE POLICY *******")
loop._update_loop_details()
loop.update_microservice_policy()

logger.info("******** ADD OPERATIONAL POLICY MINMAX *******")
added = loop.add_operational_policy(policy_type="onap.policies.controlloop.guard.
↪common.MinMax",
                                    policy_version="1.0.0")

logger.info("******** CONFIGURE OPERATIONAL POLICY MINMAX *******")
loop.add_op_policy_config(loop.add_minmax_config)

logger.info("******** ADD FREQUENCY POLICY *******")
added = loop.add_operational_policy(policy_type="onap.policies.controlloop.guard.
↪common.FrequencyLimiter",
                                    policy_version="1.0.0")

logger.info("******** CONFIGURE FREQUENCY POLICY *******")
loop.add_op_policy_config(loop.add_frequency_limiter)

logger.info("******** SUBMIT POLICIES TO PE *******")
submit = loop.act_on_loop_policy(loop.submit)

logger.info("******** CHECK POLICIES SUBMITION *******")
if submit :
    logger.info("Policies successfully submited to PE")
else:
    logger.error("An error occured while submitting the loop instance")
    exit(1)

logger.info("******** DEPLOY LOOP INSTANCE *******")
deploy = loop.deploy_microservice_to_dcae()
if deploy:
    logger.info("Loop instance %s successfully deployed on DCAE !!", LOOP_INSTANCE_
↪NAME)
```

(continues on next page)

(continued from previous page)

```
else:
    logger.error("An error occured while deploying the loop instance")
    exit(2)

logger.info("******** DELETE LOOP INSTANCE *******")
loop.delete()
```

5.3 E2E Instantiation of vFW (macro)

```
import logging
import time
import json
from uuid import uuid4
from onapsdk.aai.aai_element import AaiElement
from onapsdk.aai.cloud_infrastructure import (
    CloudRegion,
    Complex,
    Tenant
)
from onapsdk.aai.service_design_and_creation import (
    Service as AaiService
)
from onapsdk.aai.business import (
    ServiceInstance,
    VnfInstance,
    VfModuleInstance,
    ServiceSubscription,
    Customer,
    OwningEntity as AaiOwningEntity
)
from onapsdk.so.instantiation import (
    ServiceInstantiation,
    VnfInstantiation,
    InstantiationParameter,
    VnfParameters,
    VfmoduleParameters
)
from onapsdk.sdc.properties import Property
from onapsdk.sdc import SDC
from onapsdk.sdc.vendor import Vendor
from onapsdk.sdc.vsp import Vsp
from onapsdk.sdc.vf import Vf
from onapsdk.sdc.service import Service, ServiceInstantiationType
import onapsdk.constants as const
import os
from onapsdk.vid import LineOfBusiness, OwningEntity, Platform, Project

from onapsdk.cds.blueprint import Blueprint
from onapsdk.cds.data_dictionary import DataDictionary, DataDictionarySet

logger = logging.getLogger("")
logger.setLevel(logging.INFO)
logname = "./vfwcds.debug.log"
fh = logging.FileHandler(logname)
```

(continues on next page)

(continued from previous page)

```
fh_formatter = logging.Formatter('%(asctime)s %(levelname)s %(lineno)d:%(filename)s(
↪%(process)d) - %(message)s')
fh.setFormatter(fh_formatter)
logger.addHandler(fh)

################################################################################
######## CDS Design settings ###################################################
######## vFW CDS Example     ###################################################
################################################################################
# DDF Settings (dd files located in following location)
DDDIR = "resources/starter-dictionary"
DDFILE = "resources/my_dd.json"

# CBA resources (location of base CBA file)
CBAFILE = "resources/vFWCDS/CBA/CBA.zip"
ARTIFACT_LABEL = "vnfcds"
ARTIFACT_NAME = "CBA_enriched.zip"
ARTIFACT_TYPE = "CONTROLLER_BLUEPRINT_ARCHIVE"
ARTIFACT_FILE_PATH = "resources/vFWCDS/CBA/CBA_enriched.zip"
SDNC_TEMPLATE_NAME = "vFW-CDS"
SDNC_TEMPLATE_VERSION = "1.0.0"
SDNC_ARTIFACT_NAME = "vnf"

################################################################################
######## Service Design settings ###############################################
################################################################################
VENDOR = "VNFVendor"

# HEAT resources (location of zipped HEAT file)
VSPFILE = "resources/vFWCDS/HEAT/vFW/vFW.zip"
VSPNAME = "vfwcds_VS"
VFNAME = "vfwcds_VF"
SERVICENAME = "vfwcds_SERVICE"

################################################################################
######## Runtime preparation settings ##########################################
################################################################################
# Default Cloud
CLOUD_OWNER = "CloudOwner"
CLOUD_REGION = "RegionOne"

GLOBAL_CUSTOMER_ID = "generic"
CLOUD_TYPE = "openstack"
CLOUD_VERSION = "pike"
VIM_USERNAME = <user>  # FILL ME
VIM_PASSWORD = <password>  # FILL ME
VIM_SERVICE_URL = "http://<vim-url>/v3"  # FILL ME
TENANT_NAME = <tenant>  # FILL ME
TENANT_SEC_GROUP = <sec-group>  # FILL ME
COMPLEX_PHYSICAL_LOCATION_ID = "location"
COMPLEX_DATA_CENTER_CODE = "1234"


# common
OWNING_ENTITY = "Test-OE"
PROJECT = "Test-Project"
PLATFORM = "Test-Platform"
```

(continues on next page)

(continued from previous page)

```
LINE_OF_BUSINESS = "Test-BusinessLine"

SERVICE_DELETION = False

##########################################################################
######## Service Instance attributes #####################################
##########################################################################
SERVICE_INSTANCE_NAME = "vFWCDS-Instance-1"
ONAP_PRIVATE_NET = "onap-oam"    # FILL ME
ONAP_PRIVATE_SUBNET = "onap-oam-subnet" # FILL ME
PUBLIC_NET = "admin"    # FILL ME
IMAGE_NAME = "Ubuntu_1604"    # FILL ME
FLAVOR_NAME = "m1.small"     # FILL ME

logger.info("******************************")
logger.info("******** CBA Creation ********")
logger.info("******************************")

logger.info("******** Load Data Dictionary *******")
mypath = os.path.dirname(os.path.realpath(__file__))
myddpath = os.path.join(mypath, DDDIR)
myddfile = os.path.join(mypath, DDFILE)

logger.info("path: %s", myddpath)
dd_set = DataDictionarySet()
for file in os.listdir(myddpath):
    logger.info("file: %s", file)
    if file.endswith(".json"):
        with open(os.path.join(myddpath, file), "r") as dd_file:  # type file
            dd_json: dict = json.loads(dd_file.read())
            logger.info("DD: %s", dd_json)
        dd_set.add(DataDictionary(dd_json))
logger.info("DD Length: %d", dd_set.length)
dd_set.upload()

logger.info("******** Open Blueprint *******")
cbafile = os.path.join(mypath, CBAFILE)
artifactfile = os.path.join(mypath, ARTIFACT_FILE_PATH)

blueprint = Blueprint.load_from_file(cbafile)
enriched_blueprint = blueprint.enrich()  # returns enriched blueprint object
enriched_blueprint.save(artifactfile)


logger.info("******************************")
logger.info("******** SERVICE DESIGN *******")
logger.info("******************************")

logger.info("******** Onboard Vendor *******")
vendor = Vendor(name=VENDOR)
vendor.onboard()

logger.info("******** Onboard VSP *******")
vspfile = os.path.join(mypath, VSPFILE)
vsp = Vsp(name=VSPNAME, vendor=vendor, package=open(vspfile, 'rb'))
vsp.onboard()
```

(continues on next page)

(continued from previous page)

```
logger.info("******** Onboard VF *******")
vf = Vf(name=VFNAME)
vf.vsp = vsp
vf.create()

if vf.status == const.DRAFT:

    logger.info("******** Extract Artifact Data *******")
    data = open(artifactfile, 'rb').read()

    logger.info("******** Upload Artifact *******")
    vf.add_deployment_artifact(artifact_type=ARTIFACT_TYPE,
                               artifact_name=ARTIFACT_NAME,
                               artifact_label=ARTIFACT_LABEL,
                               artifact=artifactfile)

vf.onboard()

svc = Service(name=SERVICENAME,instantiation_type=ServiceInstantiationType.MACRO)
svc.create()

if svc.status == const.DRAFT:
    svc.add_resource(vf)

    logger.info("******** Set SDNC properties for VF ********")
    component = svc.get_component(vf)
    prop = component.get_property("sdnc_model_version")
    prop.value = SDNC_TEMPLATE_VERSION
    prop = component.get_property("sdnc_artifact_name")
    prop.value = SDNC_ARTIFACT_NAME
    prop = component.get_property("sdnc_model_name")
    prop.value = SDNC_TEMPLATE_NAME
    prop = component.get_property("controller_actor")
    prop.value = "CDS"
    prop = component.get_property("skip_post_instantiation_configuration")
    prop.value = False

    logger.info("******** Onboard Service *******")
    svc.checkin()
    svc.onboard()

logger.info("******** Check Service Distribution *******")
distribution_completed = False
nb_try = 0
nb_try_max = 10
while distribution_completed is False and nb_try < nb_try_max:
    distribution_completed = svc.distributed
    if distribution_completed is True:
       logger.info("Service Distribution for %s is sucessfully finished",svc.name)
       break
    logger.info("Service Distribution for %s ongoing, Wait for 60 s",svc.name)
    time.sleep(60)
    nb_try += 1

if distribution_completed is False:
    logger.error("Service Distribution for %s failed !!",svc.name)
```

(continues on next page)

(continued from previous page)

```
    exit(1)

logger.info("*******************************")
logger.info("***** RUNTIME PREPARATION *****")
logger.info("*******************************")

logger.info("******** Create Complex *******")
cmplx = Complex.create(
    physical_location_id=COMPLEX_PHYSICAL_LOCATION_ID,
    data_center_code=COMPLEX_DATA_CENTER_CODE,
    name=COMPLEX_PHYSICAL_LOCATION_ID
)

logger.info("******** Create CloudRegion *******")
cloud_region = CloudRegion.create(
    cloud_owner=CLOUD_OWNER,
    cloud_region_id=CLOUD_REGION,
    orchestration_disabled=False,
    in_maint=False,
    cloud_type=CLOUD_TYPE,
    cloud_zone="z1",
    complex_name=COMPLEX_PHYSICAL_LOCATION_ID,
    sriov_automation=False,
    owner_defined_type="t1",
    cloud_region_version=CLOUD_VERSION
)

logger.info("******** Link Complex to CloudRegion *******")
cloud_region.link_to_complex(cmplx)

logger.info("******** Add ESR Info to CloudRegion *******")
cloud_region.add_esr_system_info(
    esr_system_info_id=str(uuid4()),
    user_name=VIM_USERNAME,
    password=VIM_PASSWORD,
    system_type="VIM",
    service_url=VIM_SERVICE_URL,
    cloud_domain="Default",
    ssl_insecure=False,
    system_status="active",
    default_tenant=TENANT_NAME
)

logger.info("******** Register CloudRegion to MultiCloud *******")
cloud_region.register_to_multicloud()

logger.info("******** Check MultiCloud Registration *******")
time.sleep(60)
tenant_found = False
availability_zone_found = False
registration_completed = False
nb_try = 0
nb_try_max = 10
while registration_completed is False and nb_try < nb_try_max:
    for tenant in cloud_region.tenants:
        logger.debug("Tenant %s found in %s_%s",tenant.name,cloud_region.cloud_owner,
↪cloud_region.cloud_region_id)
```

(continues on next page)

(continued from previous page)

```
        tenant_found = True
    for az in cloud_region.availability_zones:
        logger.debug("A-Zone %s found",az.name)
        availability_zone_found = True
    if availability_zone_found and tenant_found:
        registration_completed = True
    if registration_completed is False:
        time.sleep(60)
    nb_try += 1

if registration_completed is False:
    logger.error("Registration of Cloud %s_%s failed !!",cloud_region.cloud_owner,
↪cloud_region.cloud_region_id)
    exit(1)
else:
    logger.info("Registration of Cloud %s_%s successful !!",cloud_region.cloud_owner,
↪cloud_region.cloud_region_id)

logger.info("*****************************")
logger.info("**** SERVICE INSTANTIATION ****")
logger.info("*****************************")

logger.info("******** Create Customer *******")
customer = None
for found_customer in list(Customer.get_all()):
    logger.debug("Customer %s found", found_customer.subscriber_name)
    if found_customer.subscriber_name == GLOBAL_CUSTOMER_ID:
        logger.info("Customer %s found", found_customer.subscriber_name)
        customer = found_customer
        break
if not customer:
    customer = Customer.create(GLOBAL_CUSTOMER_ID,GLOBAL_CUSTOMER_ID, "INFRA")

logger.info("******** Find Service in SDC *******")
service = None
services = Service.get_all()
for found_service in services:
    logger.debug("Service %s is found, distribution %s",found_service.name,    found_
↪service.distribution_status)
    if found_service.name == SERVICENAME:
        logger.info("Found Service %s in SDC",found_service.name)
        service = found_service
        break

if not service:
    logger.error("Service %s not found in SDC",SERVICENAME)
    exit(1)

logger.info("******** Check Service Subscription *******")
service_subscription = None
for service_sub in customer.service_subscriptions:
    logger.debug("Service subscription %s is found",service_sub.service_type)
    if service_sub.service_type == SERVICENAME:
        logger.info("Service %s subscribed",SERVICENAME)
        service_subscription = service_sub
        break
```

(continues on next page)

(continued from previous page)

```
if not service_subscription:
    logger.info("******** Subscribe Service *******")
    customer.subscribe_service(SERVICENAME)

logger.info("******** Get Tenant *******")
cloud_region = CloudRegion(cloud_owner=CLOUD_OWNER, cloud_region_id=CLOUD_REGION,
                           orchestration_disabled=True, in_maint=False)
tenant = None
for found_tenant in cloud_region.tenants:
    logger.debug("Tenant %s found in %s_%s",found_tenant.name,cloud_region.cloud_
↪owner,cloud_region.cloud_region_id)
    if found_tenant.name == TENANT_NAME:
        logger.info("Found my Tenant %s",found_tenant.name)
        tenant = found_tenant
        break

if not tenant:
    logger.error("tenant %s not found",TENANT_NAME)
    exit(1)

logger.info("******** Connect Service to Tenant *******")
service_subscription = None
for service_sub in customer.service_subscriptions:
    logger.debug("Service subscription %s is found",service_sub.service_type)
    if service_sub.service_type == SERVICENAME:
        logger.info("Service %s subscribed",SERVICENAME)
        service_subscription = service_sub
        break

if not service_subscription:
    logger.error("Service subscription %s is not found",SERVICENAME)
    exit(1)

service_subscription.link_to_cloud_region_and_tenant(cloud_region, tenant)

logger.info("******** Add Business Objects (OE, P, Pl, LoB) in VID *******")
vid_owning_entity = OwningEntity.create(OWNING_ENTITY)
vid_project = Project.create(PROJECT)
vid_platform = Platform.create(PLATFORM)
vid_line_of_business = LineOfBusiness.create(LINE_OF_BUSINESS)

logger.info("******** Add Owning Entity in AAI *******")
owning_entity = None
for oe in AaiOwningEntity.get_all():
    if oe.name == vid_owning_entity.name:
        owning_entity = oe
        break
if not owning_entity:
    logger.info("******** Owning Entity not existing: create *******")
    owning_entity = AaiOwningEntity.create(vid_owning_entity.name, str(uuid4()))

########################################################################
######## VFModule parameters ###########################################
########################################################################
vfm_base=[
   InstantiationParameter(name="sec_group", value=TENANT_SEC_GROUP),
   InstantiationParameter(name="public_net_id", value=PUBLIC_NET)
```

(continues on next page)

(continued from previous page)

```
]

vfm_vsn=[
   InstantiationParameter(name="sec_group", value=TENANT_SEC_GROUP),
   InstantiationParameter(name="public_net_id", value=PUBLIC_NET)
]

vfm_vfw=[
   InstantiationParameter(name="sec_group", value=TENANT_SEC_GROUP),
   InstantiationParameter(name="public_net_id", value=PUBLIC_NET)
]

vfm_vpkg=[
   InstantiationParameter(name="sec_group", value=TENANT_SEC_GROUP),
   InstantiationParameter(name="public_net_id", value=PUBLIC_NET)
]

base_paras=VfmoduleParameters("base_template",vfm_base)
vpkg_paras=VfmoduleParameters("vpkg",vfm_vpkg)
vsn_paras=VfmoduleParameters("vsn",vfm_vsn)
vfw_paras=VfmoduleParameters("vfw",vfm_vfw)

##########################################################################
######## VNF parameters ##################################################
##########################################################################

vnf_vfw=[
   InstantiationParameter(name="onap_private_net_id", value=ONAP_PRIVATE_NET),
   InstantiationParameter(name="onap_private_subnet_id", value=ONAP_PRIVATE_SUBNET),
   InstantiationParameter(name="pub_key", value="ssh-rsa
↪AAAAB3NzaC1yc2EAAAADAQABAAABAQDFBOB1Ea2yej68aqIQw10kEsVf+rNoxT39qrV8JvvTK2yhkniQka1t2oD9h6DlXOLM3H
↪xchLBQmqZ4AGhMIiYMfJJF+Ygy0lbgcVmT+8DH7kUUt8SAdh2rRsYFwpKANnQJyPV1dBNuTcD0OW1hEOhXnwqH28tjfb7uHJzT
↪w+YBQaIMcil"),
   InstantiationParameter(name="image_name", value=IMAGE_NAME),
   InstantiationParameter(name="flavor_name", value=FLAVOR_NAME),
   InstantiationParameter(name="sec_group", value=TENANT_SEC_GROUP),
   InstantiationParameter(name="install_script_version", value="1.4.0-SNAPSHOT"),
   InstantiationParameter(name="demo_artifacts_version", value="1.4.0-SNAPSHOT"),
   InstantiationParameter(name="cloud_env", value=CLOUD_TYPE),
   InstantiationParameter(name="public_net_id", value=PUBLIC_NET),
   InstantiationParameter(name="aic-cloud-region", value=CLOUD_REGION)
]

vnf_paras=VnfParameters("vfwcds_VF", vnf_vfw,
          [base_paras, vpkg_paras, vsn_paras, vfw_paras])

# You must define for each VNF and its vFModule the parameters,
# otherwise they stay empty.
# The matching critera are:
# - VnfParameters.name must match VNF ModelInstanceName
#   (see above "vfwcds_VF")
# - VfmoduleParameters.name must match substring in vfModule "instanceName"
#   (e.g. "vfwcds_vf0..VfwcdsVf..vsn..module-1")
logger.info("******** Instantiate Service *******")

service_instantiation = ServiceInstantiation.instantiate_macro(
    service,
```

(continues on next page)

(continued from previous page)

```
    cloud_region,
    tenant,
    customer,
    owning_entity,
    vid_project,
    vid_line_of_business,
    vid_platform,
    service_instance_name=SERVICE_INSTANCE_NAME,
    vnf_parameters=[vnf_paras]
)

if service_instantiation.wait_for_finish():
    logger.info("Success")
else:
    logger.error("Instantiation failed, check logs")
    exit(1)

service_instance = None
for se in service_subscription.service_instances:
   if se.instance_name == SERVICE_INSTANCE_NAME:
       service_instance = se
       break
if not service_instance:
    logger.error("******** Service %s instantiation failed",SERVICE_INSTANCE_NAME)
    exit(1)

if SERVICE_DELETION is False:
    logger.info("****************************************")
    logger.info("**** No Deletion requested, finished ****")
    logger.info("****************************************")
    exit(0)

logger.info("******************************")
logger.info("**** SERVICE DELETION *********")
logger.info("******************************")
time.sleep(30)

logger.info("******** Delete Service %s *******",service_instance.name)
service_deletion = service_instance.delete()

nb_try = 0
nb_try_max = 30
while not service_deletion.finished and nb_try < nb_try_max:
    logger.info("Wait for Service deletion")
    nb_try += 1
    time.sleep(10)
if service_deletion.finished:
    logger.info("Service %s deleted",service_instance.name)
else:
    logger.error("Service deletion %s failed",service_instance.name)
    exit(1)
```

5.4 E2E Instantiation of a simple Network

```
import logging
import time
from uuid import uuid4
from onapsdk.aai.aai_element import AaiElement
from onapsdk.aai.cloud_infrastructure import (
    CloudRegion,
    Complex,
    Tenant
)
from onapsdk.aai.service_design_and_creation import (
    Service as AaiService
)
from onapsdk.aai.business import (
    ServiceInstance,
    ServiceSubscription,
    Customer,
    OwningEntity as AaiOwningEntity
)
from onapsdk.so.instantiation import (
    ServiceInstantiation,
    Subnet
)
from onapsdk.sdc.service import Service
from onapsdk.sdc.vl import Vl
import onapsdk.constants as const
import os
from onapsdk.vid import LineOfBusiness, OwningEntity, Platform, Project

logger = logging.getLogger("")
logger.setLevel(logging.INFO)
fh = logging.StreamHandler()
fh_formatter = logging.Formatter('%(asctime)s %(levelname)s %(lineno)d:%(filename)s(
↪%(process)d) - %(message)s')
fh.setFormatter(fh_formatter)
logger.addHandler(fh)



# Create required A&AI resources
VL_NAME = "Generic NeutronNet"
SERVICENAME = "net_SERVICE"

GLOBAL_CUSTOMER_ID = ""  # FILL ME
COMPLEX_PHYSICAL_LOCATION_ID = ""  # FILL ME
COMPLEX_DATA_CENTER_CODE = ""  # FILL ME

CLOUD_OWNER = ""  # FILL ME
CLOUD_REGION = ""  # FILL ME

VIM_USERNAME = ""  # FILL ME
VIM_PASSWORD = ""  # FILL ME
VIM_SERVICE_URL = ""  # FILL ME

TENANT_NAME = ""  # FILL ME
OWNING_ENTITY = ""  # FILL ME
```

(continues on next page)

(continued from previous page)

```
PROJECT = ""  # FILL ME
PLATFORM = ""  # FILL ME
LINE_OF_BUSINESS = ""  # FILL ME

SERVICE_INSTANCE_NAME = "net-Instance"
SERVICE_DELETION = True

logger.info("******************************")
logger.info("******** SERVICE DESIGN *******")
logger.info("******************************")

logger.info("******** Get VL *******")
vl = Vl(VL_NAME)

logger.info("******** Onboard Service *******")
svc = Service(name=SERVICENAME, resources=[vl])
svc.onboard()

logger.info("******** Check Service Distribution *******")
distribution_completed = False
nb_try = 0
nb_try_max = 10
while distribution_completed is False and nb_try < nb_try_max:
    distribution_completed = svc.distributed
    if distribution_completed is True:
       logger.info("Service Distribution for %s is sucessfully finished",svc.name)
       break
    logger.info("Service Distribution for %s ongoing, Wait for 60 s",svc.name)
    time.sleep(60)
    nb_try += 1

if distribution_completed is False:
    logger.error("Service Distribution for %s failed !!",svc.name)
    exit(1)

logger.info("******************************")
logger.info("***** RUNTIME PREPARATION *****")
logger.info("******************************")

logger.info("******** Create Complex *******")
cmplx = Complex.create(
    physical_location_id=COMPLEX_PHYSICAL_LOCATION_ID,
    data_center_code=COMPLEX_DATA_CENTER_CODE,
    name=COMPLEX_PHYSICAL_LOCATION_ID
)

logger.info("******** Create CloudRegion *******")
cloud_region = CloudRegion.create(
    cloud_owner=CLOUD_OWNER,
    cloud_region_id=CLOUD_REGION,
    orchestration_disabled=False,
    in_maint=False,
    cloud_type="openstack",
    cloud_region_version="titanium_cloud",
    cloud_zone="z1",
    complex_name=COMPLEX_PHYSICAL_LOCATION_ID
)
```

(continues on next page)

(continued from previous page)

```
logger.info("******** Link Complex to CloudRegion *******")
cloud_region.link_to_complex(cmplx)

logger.info("******** Add ESR Info to CloudRegion *******")
cloud_region.add_esr_system_info(
    esr_system_info_id=str(uuid4()),
    user_name=VIM_USERNAME,
    password=VIM_PASSWORD,
    system_type="VIM",
    service_url=VIM_SERVICE_URL,
    cloud_domain="Default",
    ssl_insecure=False,
    system_status="active",
    default_tenant=TENANT_NAME
)

logger.info("******** Register CloudRegion to MultiCloud *******")
cloud_region.register_to_multicloud()

logger.info("******** Check MultiCloud Registration *******")
time.sleep(60)
registration_completed = False
nb_try = 0
nb_try_max = 10
while registration_completed is False and nb_try < nb_try_max:
    for tenant in cloud_region.tenants:
        logger.debug("Tenant %s found in %s_%s",tenant.name,cloud_region.cloud_owner,
↪cloud_region.cloud_region_id)
        registration_completed = True
    if registration_completed is False:
        time.sleep(60)
    nb_try += 1

if registration_completed is False:
    logger.error("Registration of Cloud %s_%s failed !!",cloud_region.cloud_owner,
↪cloud_region.cloud_region_id)
    exit(1)
else:
    logger.info("Registration of Cloud %s_%s successful !!",cloud_region.cloud_owner,
↪cloud_region.cloud_region_id)

logger.info("*****************************")
logger.info("**** SERVICE INSTANTIATION ****")
logger.info("*****************************")

logger.info("******** Create Customer *******")
customer = None
for found_customer in list(Customer.get_all()):
    logger.debug("Customer %s found", found_customer.subscriber_name)
    if found_customer.subscriber_name == GLOBAL_CUSTOMER_ID:
        logger.info("Customer %s found", found_customer.subscriber_name)
        customer = found_customer
        break
if not customer:
    customer = Customer.create(GLOBAL_CUSTOMER_ID,GLOBAL_CUSTOMER_ID, "INFRA")
```

(continues on next page)

(continued from previous page)

```
logger.info("******** Find Service in SDC *******")
service = None
services = Service.get_all()
for found_service in services:
    logger.debug("Service %s is found, distribution %s",found_service.name, found_
↪service.distribution_status)
    if found_service.name == SERVICENAME:
        logger.info("Found Service %s in SDC",found_service.name)
        service = found_service
        break

if not service:
    logger.error("Service %s not found in SDC",SERVICENAME)
    exit(1)

logger.info("******** Check Service Subscription *******")
service_subscription = None
for service_sub in customer.service_subscriptions:
    logger.debug("Service subscription %s is found",service_sub.service_type)
    if service_sub.service_type == SERVICENAME:
        logger.info("Service %s subscribed",SERVICENAME)
        service_subscription = service_sub
        break

if not service_subscription:
    logger.info("******** Subscribe Service *******")
    customer.subscribe_service(SERVICENAME)

logger.info("******** Get Tenant *******")
cloud_region = CloudRegion(cloud_owner=CLOUD_OWNER, cloud_region_id=CLOUD_REGION,
                           orchestration_disabled=True, in_maint=False)
tenant = None
for found_tenant in cloud_region.tenants:
    logger.debug("Tenant %s found in %s_%s",found_tenant.name,cloud_region.cloud_
↪owner,cloud_region.cloud_region_id)
    if found_tenant.name == TENANT_NAME:
        logger.info("Found my Tenant %s",found_tenant.name)
        tenant = found_tenant
        break

if not tenant:
    logger.error("tenant %s not found",TENANT_NAME)
    exit(1)

logger.info("******** Connect Service to Tenant *******")
service_subscription = None
for service_sub in customer.service_subscriptions:
    logger.debug("Service subscription %s is found",service_sub.service_type)
    if service_sub.service_type == SERVICENAME:
        logger.info("Service %s subscribed",SERVICENAME)
        service_subscription = service_sub
        break

if not service_subscription:
    logger.error("Service subscription %s is not found",SERVICENAME)
    exit(1)
```

(continues on next page)

(continued from previous page)

```
service_subscription.link_to_cloud_region_and_tenant(cloud_region, tenant)

logger.info("******** Add Business Objects (OE, P, Pl, LoB) in VID *******")
vid_owning_entity = OwningEntity.create(OWNING_ENTITY)
vid_project = Project.create(PROJECT)
vid_platform = Platform.create(PLATFORM)
vid_line_of_business = LineOfBusiness.create(LINE_OF_BUSINESS)

logger.info("******** Add Owning Entity in AAI *******")
owning_entity = None
for oe in AaiOwningEntity.get_all():
    if oe.name == vid_owning_entity.name:
        owning_entity = oe
        break
if not owning_entity:
    logger.info("******** Owning Entity not existing: create *******")
    owning_entity = AaiOwningEntity.create(vid_owning_entity.name, str(uuid4()))

logger.info("******** Instantiate Service *******")
service_instance = None
service_instantiation = None
for se in service_subscription.service_instances:
   if se.instance_name == SERVICE_INSTANCE_NAME:
       service_instance = se
       break
if not service_instance:
    logger.info("******** Service Instance not existing: Instantiate *******")
    # Instantiate service
    service_instantiation = ServiceInstantiation.instantiate_so_ala_carte(
        service,
        cloud_region,
        tenant,
        customer,
        owning_entity,
        vid_project,
        service_instance_name=SERVICE_INSTANCE_NAME
    )
    time.sleep(60)
else:
    logger.info("******** Service Instance already existing *******")

service_instance = None
for se in service_subscription.service_instances:
   if se.instance_name == SERVICE_INSTANCE_NAME:
       service_instance = se
       break
if not service_instance:
    logger.error("******** Service %s instantiation failed",SERVICE_INSTANCE_NAME)
    exit(1)

nb_try = 0
nb_try_max = 10
service_active = False
while service_active is False and nb_try < nb_try_max:
    if service_instance.orchestration_status == "Active":
        logger.info("******** Service Instance %s is active *******",service_instance.
↪name)
```

(continues on next page)

(continued from previous page)

```
        service_active = True
        break
    logger.info("Service %s instantiation not complete,Status:%s, wait 10s",service_
↪instance.name,service_instance.orchestration_status)
    time.sleep(10)
    nb_try += 1

if service_active is False:
    logger.error("Service %s instantiation failed",service_instance.name)
    exit(1)


logger.info("******** Get Networks in Service Model *******")
networks = service_instance.service_subscription.sdc_service.networks

logger.info("******** Create Network *******")
sn=Subnet(name="test", start_address="127.0.0.0", gateway_address="127.0.0.1")
for network in networks:
    logger.debug("Check if Network instance of class %s exist", network.name)
    network_found = False
    for network_instance in service_instance.network_instances:
        logger.debug("Network instance %s found in Service Instance ",network_intance.
↪name)
        network_found = True
    if network_found is False:
        network_instantiation = service_instance.add_network(network, vid_line_of_
↪business, vid_platform, subnets=[sn])
        network_instantiation.wait_for_finish()


if SERVICE_DELETION is False:
    logger.info("****************************************")
    logger.info("**** No Deletion requested, finished ****")
    logger.info("****************************************")
    exit(0)

logger.info("******************************")
logger.info("**** SERVICE DELETION *********")
logger.info("******************************")
time.sleep(30)

for network_instance in service_instance.network_instances:
    logger.debug("Network instance %s found in Service Instance ",network_instance.
↪name)

    logger.info("******** Delete Network %s *******",network_instance.name)
    network_deletion = network_instance.delete()
    network_deletion.wait_for_finish()

logger.info("******** Delete Service %s *******",service_instance.name)
service_deletion = service_instance.delete()
service_deletion.wait_for_finish()
```

5.5 E2E Upload of an artifact

```
import os
import logging

from onapsdk.sdc.vsp import Vsp
from onapsdk.sdc.vf import Vf
from onapsdk.sdc.service import Service

logger = logging.getLogger("")
logger.setLevel(logging.INFO)
fh = logging.StreamHandler()
fh_formatter = logging.Formatter('%(asctime)s %(levelname)s %(lineno)d:%(filename)s(
↪%(process)d) - %(message)s')
fh.setFormatter(fh_formatter)
logger.addHandler(fh)



# Create required A&AI resources
VF_NAME = "my_VF"
SERVICENAME = "artifact_SERVICE"

ARTIFACT_NAME = "clampnode"
ARTIFACT_TYPE = "DCAE_INVENTORY_BLUEPRINT"
ARTIFACT_FILE_PATH = "{os.path.dirname(os.path.abspath(__file__))}/my_ArtifactFile.
↪yaml"


logger.info("*******************************")
logger.info("******** SERVICE DESIGN *******")
logger.info("*******************************")

logger.info("******** Get VF *******")
vf = Vf(VF_NAME)
vf.onboard()

logger.info("******** Create Service *******")
svc = Service(name=SERVICENAME)
svc.create()
svc.add_resource(vf)

logger.info("******** Extract Artifact Data *******")
data = open(ARTIFACT_FILE_PATH,'rb').read()

logger.info("******** Upload Artifact *******")
svc.add_artifact_to_vf(vnf_name=VF_NAME,
                       artifact_type=ARTIFACT_TYPE,
                       artifact_name=ARTIFACT_NAME,
                       artifact=data)

logger.info("******** Distribute Service *******")
svc.checkin()
svc.certify()
svc.distribute()
```

5.6 E2E Instantiation of a simple VM without muticloud

```
import logging
import time
from uuid import uuid4
from onapsdk.aai.aai_element import AaiElement
from onapsdk.aai.cloud_infrastructure import (
    CloudRegion,
    Complex,
    Tenant
)
from onapsdk.aai.service_design_and_creation import (
    Service as AaiService
)
from onapsdk.aai.business import (
    ServiceInstance,
    VnfInstance,
    VfModuleInstance,
    ServiceSubscription,
    Customer,
    OwningEntity as AaiOwningEntity
)
from onapsdk.so.instantiation import (
    ServiceInstantiation,
    VnfInstantiation,
    VnfParameter
)
from onapsdk.sdc import SDC
from onapsdk.sdc.vendor import Vendor
from onapsdk.sdc.vsp import Vsp
from onapsdk.sdc.vf import Vf
from onapsdk.sdc.service import Service
import onapsdk.constants as const
import os
from onapsdk.vid import LineOfBusiness, OwningEntity, Platform, Project

logger = logging.getLogger("")
logger.setLevel(logging.DEBUG)
fh = logging.StreamHandler()
fh_formatter = logging.Formatter('%(asctime)s %(levelname)s %(lineno)d:%(filename)s(
↪%(process)d) - %(message)s')
fh.setFormatter(fh_formatter)
logger.addHandler(fh)

# Required A&AI resources
VSPNAME = "ubuntu16_VSP"
VFNAME = "ubuntu16_VF"
SERVICENAME = "ubuntu16_SERVICE"

# FULLY CUSTOMIZABLE VALUES
# ************************
VENDOR = "" # FILL ME
GLOBAL_CUSTOMER_ID = "" # FILL ME
SERVICE_DELETION = True # True|False

COMPLEX_PHYSICAL_LOCATION_ID = ""  # FILL ME
COMPLEX_DATA_CENTER_CODE = ""  # FILL ME
```

(continues on next page)

(continued from previous page)

```
CLOUD_OWNER = ""  # FILL ME

OWNING_ENTITY = ""  # FILL ME
PROJECT = ""  # FILL ME
PLATFORM = ""  # FILL ME
LINE_OF_BUSINESS = ""  # FILL ME

SERVICE_INSTANCE_NAME = "" # FILL ME

AVAILABILITY_ZONE_NAME = "" # FILL ME
AVAILABILITY_ZONE_HYPERVISOR_TYPE = "" # FILL ME


# FILL ME with your INFRA values
# ******************************
# ubuntu16.zip file path including the heat and env files
VSPFILE_PATH = "" # FILL ME

VIM_USERNAME = ""  # FILL ME
VIM_PASSWORD = ""  # FILL ME
VIM_SERVICE_URL = ""  # FILL ME

TENANT_NAME = ""  # FILL ME
TENANT_ID = "" # FILL ME

CLOUD_REGION = ""  # Shall be defined in Openstack


#␣
↪*********************************************************************************************************
logger.info("*******************************")
logger.info("******** SERVICE DESIGN *******")
logger.info("*******************************")

logger.info("******** Onboard Vendor *******")
vendor = Vendor(name=VENDOR)
vendor.onboard()

logger.info("******** Onboard VSP *******")
vsp = Vsp(name=VSPNAME, vendor=vendor, package=open(VSPFILE_PATH, 'rb'))
vsp.onboard()

logger.info("******** Onboard VF *******")
vf = Vf(name=VFNAME)
vf.vsp = vsp
vf.onboard()

logger.info("******** Onboard Service *******")
svc = Service(name=SERVICENAME, resources=[vf])
svc.onboard()

logger.info("******** Check Service Distribution *******")
distribution_completed = False
nb_try = 0
nb_try_max = 10
while distribution_completed is False and nb_try < nb_try_max:
```

(continues on next page)

(continued from previous page)

```
    distribution_completed = svc.distributed
    if distribution_completed is True:
       logger.info("Service Distribution for %s is sucessfully finished",svc.name)
       break
    logger.info("Service Distribution for %s ongoing, Wait for 60 s",svc.name)
    time.sleep(60)
    nb_try += 1

if distribution_completed is False:
    logger.error("Service Distribution for %s failed !!",svc.name)
    exit(1)

logger.info("*******************************")
logger.info("***** RUNTIME PREPARATION *****")
logger.info("*******************************")

logger.info("******** Create Complex *******")
cmplx = Complex.create(
    physical_location_id=COMPLEX_PHYSICAL_LOCATION_ID,
    data_center_code=COMPLEX_DATA_CENTER_CODE,
    name=COMPLEX_PHYSICAL_LOCATION_ID
)

logger.info("******** Create CloudRegion *******")
# Note for non multicloud instanciation, cloud_region_version shall be set to␣
↪openstack
# versus
cloud_region = CloudRegion.create(
    cloud_owner=CLOUD_OWNER,
    cloud_region_id=CLOUD_REGION,
    orchestration_disabled=False,
    in_maint=False,
    cloud_type="openstack",
    cloud_region_version="openstack",
    cloud_zone="z1",
    complex_name=COMPLEX_PHYSICAL_LOCATION_ID
)

logger.info("******** Link Complex to CloudRegion *******")
cloud_region.link_to_complex(cmplx)

logger.info("******** Add ESR Info to CloudRegion *******")
cloud_region.add_esr_system_info(
    esr_system_info_id=str(uuid4()),
    user_name=VIM_USERNAME,
    password=VIM_PASSWORD,
    system_type="VIM",
    service_url=VIM_SERVICE_URL,
    cloud_domain="Default",
    ssl_insecure=False,
    system_status="active",
    default_tenant=TENANT_NAME
)

logger.info("*******************************")
logger.info("**** SERVICE INSTANTIATION ****")
logger.info("*******************************")
```

(continues on next page)

(continued from previous page)

```
logger.info("******** Create Customer *******")
customer = None
for found_customer in list(Customer.get_all()):
    logger.debug("Customer %s found", found_customer.subscriber_name)
    if found_customer.subscriber_name == GLOBAL_CUSTOMER_ID:
        logger.info("Customer %s found", found_customer.subscriber_name)
        customer = found_customer
        break
if not customer:
    customer = Customer.create(GLOBAL_CUSTOMER_ID,GLOBAL_CUSTOMER_ID, "INFRA")

logger.info("******** Find Service in SDC *******")
service = None
services = Service.get_all()
for found_service in services:
    logger.debug("Service %s is found, distribution %s",found_service.name, found_
↪service.distribution_status)
    if found_service.name == SERVICENAME:
        logger.info("Found Service %s in SDC",found_service.name)
        service = found_service
        break

if not service:
    logger.error("Service %s not found in SDC",SERVICENAME)
    exit(1)

logger.info("******** Check Service Subscription *******")
service_subscription = None
for service_sub in customer.service_subscriptions:
    logger.debug("Service subscription %s is found",service_sub.service_type)
    if service_sub.service_type == SERVICENAME:
        logger.info("Service %s subscribed",SERVICENAME)
        service_subscription = service_sub
        break

if not service_subscription:
    logger.info("******** Subscribe Service *******")
    customer.subscribe_service(SERVICENAME)

logger.info("******** Get Tenant *******")
cloud_region = CloudRegion(cloud_owner=CLOUD_OWNER, cloud_region_id=CLOUD_REGION,
                           orchestration_disabled=True, in_maint=False)
 try:
     tenant: Tenant = cloud_region.get_tenant(settings.TENANT_ID)
 except ValueError:
     logger.warning("Impossible to retrieve the Specificed Tenant")
     logger.debug("If no multicloud selected, add the tenant")
     cloud_region.add_tenant(
         tenant_id=settings.TENANT_ID,
         tenant_name=settings.TENANT_NAME)

 # be sure that an availability zone has been created
 # if not, create it
 try:
     cloud_region.get_availability_zone_by_name(
         settings.AVAILABILITY_ZONE_NAME)
```

(continues on next page)

(continued from previous page)

```
 except ValueError:
     cloud_region.add_availability_zone(
         settings.AVAILABILITY_ZONE_NAME,
         settings.AVAILABILITY_ZONE_HYPERVISOR_TYPE)

logger.info("******** Connect Service to Tenant *******")
service_subscription = None
for service_sub in customer.service_subscriptions:
    logger.debug("Service subscription %s is found",service_sub.service_type)
    if service_sub.service_type == SERVICENAME:
        logger.info("Service %s subscribed",SERVICENAME)
        service_subscription = service_sub
        break

if not service_subscription:
    logger.error("Service subscription %s is not found",SERVICENAME)
    exit(1)

service_subscription.link_to_cloud_region_and_tenant(cloud_region, tenant)

logger.info("******** Add Business Objects (OE, P, Pl, LoB) in VID *******")
vid_owning_entity = OwningEntity.create(OWNING_ENTITY)
vid_project = Project.create(PROJECT)
vid_platform = Platform.create(PLATFORM)
vid_line_of_business = LineOfBusiness.create(LINE_OF_BUSINESS)

logger.info("******** Add Owning Entity in AAI *******")
owning_entity = None
for oe in AaiOwningEntity.get_all():
    if oe.name == vid_owning_entity.name:
        owning_entity = oe
        break
if not owning_entity:
    logger.info("******** Owning Entity not existing: create *******")
    owning_entity = AaiOwningEntity.create(vid_owning_entity.name, str(uuid4()))

logger.info("******** Instantiate Service *******")
service_instance = None
service_instantiation = None
for se in service_subscription.service_instances:
   if se.instance_name == SERVICE_INSTANCE_NAME:
       service_instance = se
       break
if not service_instance:
    logger.info("******** Service Instance not existing: Instantiate *******")
    # Instantiate service
    service_instantiation = ServiceInstantiation.instantiate_so_ala_carte(
        service,
        cloud_region,
        tenant,
        customer,
        owning_entity,
        vid_project,
        service_instance_name=SERVICE_INSTANCE_NAME
    )
    time.sleep(60)
else:
```

(continues on next page)

(continued from previous page)

```
    logger.info("******** Service Instance already existing *******")

service_instance = None
for se in service_subscription.service_instances:
   if se.instance_name == SERVICE_INSTANCE_NAME:
       service_instance = se
       break
if not service_instance:
    logger.error("******** Service %s instantiation failed",SERVICE_INSTANCE_NAME)
    exit(1)

nb_try = 0
nb_try_max = 10
service_active = False
while service_active is False and nb_try < nb_try_max:
    if service_instance.orchestration_status == "Active":
       logger.info("******** Service Instance %s is active *******",service_instance.
↪name)
       service_active = True
       break
    logger.info("Service %s instantiation not complete,Status:%s, wait 10s",service_
↪instance.name,service_instance.orchestration_status)
    time.sleep(10)
    nb_try += 1

if service_active is False:
    logger.error("Service %s instantiation failed",service_instance.name)
    exit(1)


logger.info("******** Get VNFs in Service Model *******")
vnfs = service_instance.service_subscription.sdc_service.vnfs

logger.info("******** Create VNFs *******")
for vnf in vnfs:
    logger.debug("Check if VNF instance of class %s exist", vnf.name)
    vnf_found = False
    for vnf_instance in service_instance.vnf_instances:
        logger.debug("VNF instance %s found in Service Instance ",vnf_instance.name)
        vnf_found = True
    if vnf_found is False:
        vnf_instantiation = service_instance.add_vnf(vnf, vid_line_of_business, vid_
↪platform)
        while not vnf_instantiation.finished:
            print("Wait for VNF %s instantiation",vnf.name)
            time.sleep(10)


for vnf_instance in service_instance.vnf_instances:
    logger.debug("VNF instance %s found in Service Instance ",vnf_instance.name)
    logger.info("******** Get VfModules in VNF Model *******")
    logger.info("******** Check VF Modules *******")
    vf_module = vnf_instance.vnf.vf_module

    logger.info("******** Create VF Module %s *******",vf_module.name)

    for vf_module in vnf_instance.vnf.vf_modules:
```

(continues on next page)

(continued from previous page)

```
      vf_module_instantiation = vnf_instance.add_vf_module(
        vf_module,
        cloud_region,tenant,
        SERVICE_INSTANCE_NAME,
        vnf_parameters=[])
      nb_try = 0
      nb_try_max = 30
      while not vf_module_instantiation.finished and nb_try < nb_try_max:
          logger.info("Wait for vf module instantiation")
          nb_try += 1
          time.sleep(10)
      if vf_module_instantiation.finished:
          logger.info("VfModule %s instantiated",vf_module.name)
      else:
          logger.error("VfModule instantiation %s failed",vf_module.name)

if SERVICE_DELETION is False:
    logger.info("****************************************")
    logger.info("**** No Deletion requested, finished ****")
    logger.info("****************************************")
    exit(0)

logger.info("******************************")
logger.info("**** SERVICE DELETION *********")
logger.info("******************************")
time.sleep(30)

for vnf_instance in service_instance.vnf_instances:
    logger.debug("VNF instance %s found in Service Instance ",vnf_instance.name)
    logger.info("******** Get VF Modules *******")
    for vf_module in vnf_instance.vf_modules:
        logger.info("******** Delete VF Module %s *******",vf_module.name)
        vf_module_deletion = vf_module.delete()

        nb_try = 0
        nb_try_max = 30
        while not vf_module_deletion.finished and nb_try < nb_try_max:
            logger.info("Wait for vf module deletion")
            nb_try += 1
            time.sleep(10)
        if vf_module_deletion.finished:
            logger.info("VfModule %s deleted",vf_module.name)
        else:
            logger.error("VfModule deletion %s failed",vf_module.name)
            exit(1)

    logger.info("******** Delete VNF %s *******",vnf_instance.name)
    vnf_deletion = vnf_instance.delete()

    nb_try = 0
    nb_try_max = 30
    while not vnf_deletion.finished and nb_try < nb_try_max:
        logger.info("Wait for vnf deletion")
        nb_try += 1
        time.sleep(10)
    if vnf_deletion.finished:
        logger.info("VNF %s deleted",vnf_instance.name)
```

(continues on next page)

(continued from previous page)

```
    else:
        logger.error("VNF deletion %s failed",vnf_instance.name)
        exit(1)

logger.info("******** Delete Service %s *******",service_instance.name)
service_deletion = service_instance.delete()

nb_try = 0
nb_try_max = 30
while not service_deletion.finished and nb_try < nb_try_max:
    logger.info("Wait for Service deletion")
    nb_try += 1
    time.sleep(10)
if service_deletion.finished:
    logger.info("Service %s deleted",service_instance.name)
else:
    logger.error("Service deletion %s failed",service_instance.name)
exit(1)
```

5.7 E2E msb k8s plugin usage

```
import logging
import os

from onapsdk.msb.k8s import (
    Definition,
    Instance,
    ConnectivityInfo)

logger = logging.getLogger("")
logger.setLevel(logging.DEBUG)
fh = logging.StreamHandler()
fh_formatter = logging.Formatter('%(asctime)s %(levelname)s %(lineno)d:%(filename)s(
↪%(process)d) - %(message)s')
fh.setFormatter(fh_formatter)
logger.addHandler(fh)

RB_NAME = "test_definition"
RB_VERSION = "ver_1"
DEFINITION_ARTIFACT_PATH = "artifacts\\vault-consul-dev.tar.gz"  # FILL ME
PROFILE_NAME = "test-profile"
PROFILE_NAMESPACE = "test"
PROFILE_K8S_VERSION = "1.0"
PROFILE_ARTIFACT_PATH = "artifacts\\profile.tar.gz"  # FILL ME
CLOUD_REGION_ID = "k8s_region_test"  # FILL ME
CLOUD_OWNER = "CloudOwner"
KUBECONFIG_PATH = "artifacts\\kubeconfig"  # FILL ME
MYPATH = os.path.dirname(os.path.realpath(__file__))

######## Create new Definition ################################################
definition = Definition.create(RB_NAME, RB_VERSION)

######## Upload artifact for created definition ###########################
definition_artifact_file = os.path.join(MYPATH, DEFINITION_ARTIFACT_PATH)
```

(continues on next page)

(continued from previous page)

```
definition.upload_artifact(open(definition_artifact_file, 'rb').read())

######## Get one Definition ###################################################
check_definition = Definition.get_definition_by_name_version(RB_NAME,
                                                             RB_VERSION)

######## Get all Definitions ##################################################
definitions = list(Definition.get_all())

######## Create profile for Definition ########################################
profile = definition.create_profile(PROFILE_NAME,
                                    PROFILE_NAMESPACE,
                                    PROFILE_K8S_VERSION)

######## Upload artifact for created profile ##################################
profile_artifact_file = os.path.join(MYPATH, PROFILE_ARTIFACT_PATH)
profile.upload_artifact(open(profile_artifact_file, 'rb').read())

######## Get one Profile ######################################################
check_profile = definition.get_profile_by_name(PROFILE_NAME)

######## Get all Profiles #####################################################
profiles = list(definition.get_all_profiles())

######## Create Connectivity Info #############################################
kubeconfig_file = os.path.join(MYPATH, KUBECONFIG_PATH)
conninfo = ConnectivityInfo.create(CLOUD_REGION_ID,
                                   CLOUD_OWNER,
                                   open(kubeconfig_file, 'rb').read())

######## Instantiate Profile ##################################################
instance = Instance.create(CLOUD_REGION_ID,
                           profile.profile_name,
                           definition.rb_name,
                           definition.rb_version)

######## Get Instance by ID ###################################################
check_instance = Instance.get_by_id(instance.instance_id)

######## Get all Instances ####################################################
instances = list(Instance.get_all())

######## Delete Instance ######################################################
instance.delete()

######## Check instance deletion ##############################################
instances = list(Instance.get_all())

######## Delete Connectivity Info #############################################
conninfo.delete()

######## Delete Profile #######################################################
profile.delete()

######## Delete Definition ####################################################
definition.delete()
```

CHAPTER

SIX

DEVELOPMENT

6.1 Setting up development environment

Before you start, ensure you have Python installation in version 3.7 or higher. Please see the official Python documentation in case you have to upgrade or install certain Python version.

Clone the project. Inside the project folder create a new virtual environment and activate it:

```
$ python -m venv env
$ source env/bin/activate
```

On Windows, activate by executing the following:

```
$ .\env\Scripts\activate
```

When your virtual environment is ready, install required dependencies:

```
$ pip install -r requirements.txt
```

6.2 Developing

To use library functions directly from the source code, execute the following to point to the source folder in *PYTHONPATH* variable and run the interpreter:

```
$ PYTHONPATH=$PYTHONPATH:src/ python
```

On Windows:

```
$ $env:PYTHONPATH='src\';python
```

Verify that packages are accessible:

```
>>> import onapsdk
```

You can then start working with library functions as needed.

6.3 New ONAP component package

When you create a new ONAP component package and wants to use Jinja templates you need to create *templates* directory to store them in a newly created package. Furthermore you need to add a *PackageLoader* in *utils.jinja* module.

6.4 Testing

Install tox:

```
$ pip install tox
```

To run all unit test, lint and docstyle checks, inside the project folder simply execute *tox*:

```
$ tox
```

Please note that the above runs unit tests on all major versions of Python available on your OS (3.7, 3.8, 3.9). To limit execution to only specific version of Python Interpreter, use the following example:

```
$ tox -e py37
```

6.5 Integration testing

It is possible to run integration tests using mock-servers project.

Make sure Docker Compose is available on your system. Install required dependencies:

```
$ pip install pytest mock
```

Go to *integration_tests/* directory and execute:

```
$ docker-compose up
```

Please note that *docker-compose* attempts to create subnet 172.20.0.0/24, so it can not be run if the scope is already allocated. Also, containers are not reachable by their IP addresses on Windows host since Docker for Windows does not support bridged network interface for Linux containers. For reference, please see Docker docs.

Once containers are running, execute the following in the project's directory:

```
$ PYTHONPATH=$PYTHONPATH:integration_tests/:src/ ONAP_PYTHON_SDK_SETTINGS="local_urls
↪" pytest -c /dev/null --verbose --junitxml=pytest-integration.xml integration_tests
```

Please make sure all the test are passing before creating merge request.

CHAPTER

SEVEN

ARCHITECTURE

CHAPTER

EIGHT

ONAPSDK

8.1 onapsdk package

8.1.1 Subpackages

onapsdk.aai package

Subpackages

onapsdk.aai.business package

Submodules

onapsdk.aai.business.customer module

AAI business module.

class onapsdk.aai.business.customer.**Customer**(*global_customer_id*, *subscriber_name*, *subscriber_type*, *resource_version=None*)

Bases: *onapsdk.aai.aai_element.AaiResource*

Customer class.

classmethod create(*global_customer_id*, *subscriber_name*, *subscriber_type*, *service_subscriptions=None*)

Create customer.

Parameters

- **global_customer_id** (*str*) – Global customer id used across ONAP to uniquely identify customer.
- **subscriber_name** (*str*) – Subscriber name, an alternate way to retrieve a customer.
- **subscriber_type** (*str*) – Subscriber type, a way to provide VID with only the INFRA customers.
- **service_subscriptions** (*Optional[Iterable[str]], optional*) – Iterable of service subscription names should be created for newly created customer. Defaults to None.

Returns Customer object.

Return type *Customer*

delete()
Delete customer.

Sends request to A&AI to delete customer object.

Return type None

classmethod get_all(*global_customer_id=None*, *subscriber_name=None*, *subscriber_type=None*)
Get all customers.

Call an API to retrieve all customers. It can be filtered by global-customer-id, subscriber-name and/or subsriber-type.

Parameters

- **global_customer_id** (*str*) – global-customer-id to filer customers by. Defaults to None.
- **subscriber_name** (*str*) – subscriber-name to filter customers by. Defaults to None.
- **subscriber_type** (*str*) – subscriber-type to filter customers by. Defaults to None.

Return type Iterator[*Customer*]

classmethod get_all_url()
Return an url to get all customers.

Returns URL to get all customers

Return type str

classmethod get_by_global_customer_id(*global_customer_id*)
Get customer by it's global customer id.

Parameters **global_customer_id** (*str*) – global customer ID

Returns Customer with given global_customer_id

Return type *Customer*

get_service_subscription_by_service_type(*service_type*)
Get subscribed service by service type.

Call a request to get service subscriptions filtered by service-type parameter.

Parameters **service_type** (*str*) – Service type

Returns Service subscription

Return type *ServiceSubscription*

property service_subscriptions
Service subscriptions of customer resource.

Yields *ServiceSubscription* – ServiceSubscription object

Return type Iterator[*ServiceSubscription*]

subscribe_service(*service_type*)
Create SDC Service subscription.

If service subscription with given service_type already exists it won't create a new resource but use the existing one.

Parameters **service_type** (*str*) – Value defined by orchestration to identify this service across ONAP.

Return type *ServiceSubscription*

property url
Return customer object url.

Unique url address to get customer's data.

Returns Customer object url

Return type str

class onapsdk.aai.business.customer.**ServiceSubscription**(*customer*, *service_type*, *resource_version*)
Bases: *onapsdk.aai.aai_element.AaiResource*

Service subscription class.

property cloud_region
Cloud region associated with service subscription.

IT'S DEPRECATED! *cloud_regions* parameter SHOULD BE USED

Raises ***ParameterError*** – Service subscription has no associated cloud region.

Returns CloudRegion object

Return type *CloudRegion*

property cloud_regions
Cloud regions associated with service subscription.

Yields *CloudRegion* – CloudRegion object

Return type Iterator[*CloudRegion*]

classmethod create_from_api_response(*api_response*, *customer*)
Create service subscription using API response dict.

Returns ServiceSubscription object.

Return type *ServiceSubscription*

customer: ***onapsdk.aai.business.customer.Customer***

classmethod get_all_url(*customer*)
Return url to get all customers.

Returns Url to get all customers

Return type str

get_service_instance_by_id(*service_instance_id*)
Get service instance using it's ID.

Parameters **service_instance_id** (*str*) – ID of the service instance

Returns ServiceInstance object

Return type *ServiceInstance*

get_service_instance_by_name(*service_instance_name*)
Get service instance using it's name.

Parameters **service_instance_name** (*str*) – Name of the service instance

Returns ServiceInstance object

Return type *ServiceInstance*

link_to_cloud_region_and_tenant(*cloud_region*, *tenant*)
Create relationship between object and cloud region with tenant.

Parameters

- **cloud_region** (CloudRegion) – Cloud region to link to
- **tenant** (Tenant) – Cloud region tenant to link to

Return type None

resource_version: str

property service_instances
Service instances.

Yields *Iterator[ServiceInstance]* – Service instance

Return type Iterator[*ServiceInstance*]

service_type: str

property tenant
Tenant associated with service subscription.

IT'S DEPRECATED! *tenants* parameter SHOULD BE USED

Raises ***ParameterError*** – Service subscription has no associated tenants

Returns Tenant object

Return type *Tenant*

property tenant_relationships
Tenant related relationships.

Iterate through relationships and get related to tenant.

Yields *Relationship* – Relationship related to tenant.

Return type Iterator[*Relationship*]

property tenants
Tenants associated with service subscription.

Yields *Tenant* – Tenant object

Return type Iterator[ForwardRef]

property url
Cloud region object url.

URL used to call CloudRegion A&AI API

Returns CloudRegion object url

Return type str

class onapsdk.aai.business.customer.**ServiceSubscriptionCloudRegionTenantData**(*cloud_owner=None*, *cloud_region_id=None*, *tenant_id=None*)

Bases: object

Dataclass to store cloud regions and tenants data for service subscription.

cloud_owner: *str* **= None**

cloud_region_id: *str* **= None**

tenant_id: *str* **= None**

onapsdk.aai.business.instance module

Base instance module.

class onapsdk.aai.business.instance.**Instance**(*resource_version=None*, *model_invariant_id=None*, *model_version_id=None*)

Bases: *onapsdk.aai.aai_element.AaiResource*, abc.ABC

Abstract instance class.

abstract delete(*a_la_carte=True*)

Create instance deletion request.

Send request to delete instance

Parameters **a_la_carte** (*boolean*) – deletion mode

Returns Deletion request

Return type *DeletionRequest*

onapsdk.aai.business.line_of_business module

A&AI line of business module.

class onapsdk.aai.business.line_of_business.**LineOfBusiness**(*name*, *resource_version*)

Bases: *onapsdk.aai.aai_element.AaiResource*

Line of business class.

classmethod create(*name*)

Create line of business A&AI resource.

Parameters **name** (*str*) – line of business name

Returns Created LineOfBusiness object

Return type *LineOfBusiness*

classmethod get_all()

Get all line of business.

Yields *LineOfBusiness* – LineOfBusiness object

Return type Iterator[*LineOfBusiness*]

classmethod get_all_url()

Return url to get all lines of business.

Returns Url to get all lines of business

Return type str

classmethod get_by_name(*name*)

Get line of business resource by it's name.

Raises *ResourceNotFound* – Line of business requested by a name does not exist.

Returns Line of business requested by a name.

Return type *LineOfBusiness*

property url
Line of business's url.

Returns Resource's url

Return type str

onapsdk.aai.business.network module

Network instance module.

class onapsdk.aai.business.network.**NetworkInstance**(*service_instance*, *network_id*, *is_bound_to_vpn*, *is_provider_network*, *is_shared_network*, *is_external_network*, *network_name=None*, *network_type=None*, *network_role=None*, *network_technology=None*, *neutron_network_id=None*, *service_id=None*, *network_role_instance=None*, *resource_version=None*, *orchestration_status=None*, *heat_stack_id=None*, *mso_catalog_key=None*, *model_invariant_id=None*, *contrail_network_fqdn=None*, *persona_model_version=None*, *model_version_id=None*, *model_customization_id=None*, *widget_model_id=None*, *physical_network_name=None*, *widget_model_version=None*, *selflink=None*, *operational_status=None*, *is_trunked=None*)

Bases: *onapsdk.aai.business.instance.Instance*

Network instance class.

classmethod create_from_api_response(*api_response*, *service_instance*)
Create network instance object using HTTP API response dictionary.

Parameters

- **api_response** (*dict*) – A&AI API response dictionary
- **service_instance** (ServiceInstance) – Service instance with which network is related

Returns VnfInstance object

Return type *VnfInstance*

delete(*a_la_carte=True*)
Create network deletion request.

Send request to delete network instance

Parameters **a_la_carte** (*boolean*) – deletion mode

Returns Deletion request

Return type *NetworkDeletionRequest*

classmethod get_all_url()
Return url to get all networks.

Returns Url to get all networks

Return type str

property url
Network instance url.

Returns NetworkInstance url

Return type str

onapsdk.aai.business.owning_entity module

A&AI owning entity module.

class onapsdk.aai.business.owning_entity.**OwningEntity**(*name*, *owning_entity_id*, *resource_version*)
Bases: *onapsdk.aai.aai_element.AaiResource*

Owning entity class.

classmethod create(*name*, *owning_entity_id=None*)
Create owning entity A&AI resource.

Parameters

- **name** (*str*) – owning entity name
- **owning_entity_id** (*str*) – owning entity ID. Defaults to None.

Returns Created OwningEntity object

Return type *OwningEntity*

classmethod get_all()
Get all owning entities.

Yields *OwningEntity* – OwningEntity object

Return type Iterator[*OwningEntity*]

classmethod get_all_url()
Return url to get all owning entities.

Returns Url to get all owning entities

Return type str

classmethod get_by_owning_entity_id(*owning_entity_id*)
Get owning entity by it's ID.

Parameters **owning_entity_id** (*str*) – owning entity object id

Returns OwningEntity object

Return type *OwningEntity*

classmethod get_by_owning_entity_name(*owning_entity_name*)
Get owning entity resource by it's name.

Raises ***ResourceNotFound*** – Owning entity requested by a name does not exist.

Returns Owning entity requested by a name.

Return type *OwningEntity*

property url
Owning entity object url.

Returns Url

Return type str

onapsdk.aai.business.platform module

A&AI platform module.

class onapsdk.aai.business.platform.**Platform**(*name*, *resource_version*)
Bases: *onapsdk.aai.aai_element.AaiResource*

Platform class.

classmethod create(*name*)
Create platform A&AI resource.

Parameters **name** (*str*) – platform name

Returns Created Platform object

Return type *Platform*

classmethod get_all()
Get all platform.

Yields *Platform* – Platform object

Return type Iterator[*Platform*]

classmethod get_all_url()
Return url to get all platforms.

Returns Url to get all platforms

Return type str

classmethod get_by_name(*name*)
Get platform resource by it's name.

Raises ***ResourceNotFound*** – Platform requested by a name does not exist.

Returns Platform requested by a name.

Return type *Platform*

property url
Platform's url.

Returns Resource's url

Return type str

onapsdk.aai.business.pnf module

Pnf instance module.

class onapsdk.aai.business.pnf.**PnfInstance**(*service_instance*, *pnf_name*, *in_maint*, *self-link=None*, *pnf_id=None*, *equip_type=None*, *equip_vendor=None*, *equip_model=None*, *management_option=None*, *orchestration_status=None*, *ipaddress_v4_oam=None*, *sw_version=None*, *frame_id=None*, *serial_number=None*, *ipaddress_v4_loopback_0=None*, *ipaddress_v6_loopback_0=None*, *ipaddress_v4_aim=None*, *ipaddress_v6_aim=None*, *ipaddress_v6_oam=None*, *inv_status=None*, *resource_version=None*, *prov_status=None*, *nf_role=None*, *admin_status=None*, *operational_status=None*, *model_customization_id=None*, *model_invariant_id=None*, *model_version_id=None*, *pnf_ipv4_address=None*, *pnf_ipv6_address=None*)

Bases: *onapsdk.aai.business.instance.Instance*

Pnf instance class.

classmethod create_from_api_response(*api_response*, *service_instance*)

Create pnf instance object using HTTP API response dictionary.

Parameters

- **api_response** (*dict*) – A&AI API response dictionary
- **service_instance** (ServiceInstance) – Service instance with which network is related

Returns PnfInstance object

Return type *PnfInstance*

delete(*a_la_carte=True*)

Delete Pnf instance.

PNF deletion it's just A&AI resource deletion. That's difference between another instances. You don't have to wait for that task finish, because it's not async task.

Return type None

classmethod get_all()

Get all PNF instances.

Yields *PnfInstance* – Pnf instance

Return type Iterator[*PnfInstance*]

classmethod get_all_url()
Return an url to get all pnfs.

Returns Url to get all pnfs

Return type str

property pnf
Pnf associated with that pnf instance.

Raises ***ResourceNotFound*** – Could not find PNF for that PNF instance

Returns Pnf object associated with Pnf instance

Return type *Pnf*

property url
Network instance url.

Returns NetworkInstance url

Return type str

onapsdk.aai.business.project module

A&AI project module.

class onapsdk.aai.business.project.**Project**(*name*, *resource_version*)
Bases: *onapsdk.aai.aai_element.AaiResource*

Project class.

classmethod create(*name*)
Create project A&AI resource.

Parameters **name** (*str*) – project name

Returns Created Project object

Return type *Project*

classmethod get_all()
Get all project.

Yields *Project* – Project object

Return type Iterator[*Project*]

classmethod get_all_url()
Return url to get all projects.

Returns Url to get all projects

Return type str

classmethod get_by_name(*name*)
Get project resource by it's name.

Raises ***ResourceNotFound*** – Project requested by a name does not exist.

Returns Project requested by a name.

Return type *Project*

property url
Project's url.

Returns Resource's url

Return type str

onapsdk.aai.business.service module

Service instance module.

class onapsdk.aai.business.service.**ServiceInstance**(*service_subscription*, *instance_id*, *instance_name=None*, *service_type=None*, *service_role=None*, *environment_context=None*, *workload_context=None*, *created_at=None*, *updated_at=None*, *resource_version=None*, *description=None*, *model_invariant_id=None*, *model_version_id=None*, *persona_model_version=None*, *widget_model_id=None*, *widget_model_version=None*, *bandwith_total=None*, *vhn_portal_url=None*, *service_instance_location_id=None*, *selflink=None*, *orchestration_status=None*, *input_parameters=None*)

Bases: *onapsdk.aai.business.instance.Instance*

Service instanve class.

property active

Information if service instance's orchestration status is active.

Return type bool

add_network(*network*, *line_of_business*, *platform*, *cloud_region=None*, *tenant=None*, *network_instance_name=None*, *subnets=None*)

Add network into service instance.

Instantiate vl.

Parameters

- **network** (Network) – Network from service configuration to instantiate
- **line_of_business** (LineOfBusiness) – LineOfBusiness to use in instantiation request
- **platform** (Platform) – Platform to use in instantiation request
- **cloud_region** (CloudRegion, *optional*) – Cloud region to use in instantiation request. Defaults to None. THAT PROPERTY WILL BE REQUIRED IN ONE OF THE FUTURE RELEASE. REFACTOR YOUR CODE TO USE IT!.
- **tenant** (Tenant, *optional*) – Tenant to use in instnatiation request. Defaults to None. THAT PROPERTY WILL BE REQUIRED IN ONE OF THE FUTURE RELEASE. REFACTOR YOUR CODE TO USE IT!.

- **network_instance_name** (*str, optional*) – Network instantion name. If no value is provided it's going to be "Python_ONAP_SDK_network_instance_{str(uuid4())}". Defaults to None.

Raises ***StatusError*** – Service orchestration status is not "Active"

Returns NetworkInstantiation request object

Return type *NetworkInstantiation*

add_vnf(*vnf*, *line_of_business*, *platform*, *cloud_region=None*, *tenant=None*, *vnf_instance_name=None*, *vnf_parameters=None*, *so_vnf=None*, *a_la_carte=True*)

Add vnf into service instance.

Instantiate VNF.

Parameters

- **vnf** (Vnf) – Vnf from service configuration to instantiate
- **line_of_business** (LineOfBusiness) – LineOfBusiness to use in instantiation request
- **platform** (Platform) – Platform to use in instantiation request
- **cloud_region** (CloudRegion, *optional*) – Cloud region to use in instantiation request. Defaults to None. THAT PROPERTY WILL BE REQUIRED IN ONE OF THE FUTURE RELEASE. REFACTOR YOUR CODE TO USE IT!.
- **tenant** (Tenant, *optional*) – Tenant to use in instnatiation request. Defaults to None. THAT PROPERTY WILL BE REQUIRED IN ONE OF THE FUTURE RELEASE. REFACTOR YOUR CODE TO USE IT!.
- **vnf_instance_name** (*str, optional*) – VNF instantion name. If no value is provided it's going to be "Python_ONAP_SDK_vnf_instance_{str(uuid4())}". Defaults to None.
- **vnf_parameters** (*Iterable[InstantiationParameter], optional*) – InstantiationParameter to be passed as "userParams". Defaults to None.
- **so_vnf** (SoServiceVnf) – (SoServiceVnf, optional): object with vnf instance parameters. Defaults to None.
- **a_la_carte** (*bool*) – instantiation type for vnf. Defaults to True.

Raises ***StatusError*** – Service orchestration status is not "Active".

Returns VnfInstantiation request object

Return type *VnfInstantiation*

classmethod create(*service_subscription*, *instance_id*, *instance_name=None*, *service_type=None*, *service_role=None*, *environment_context=None*, *workload_context=None*, *created_at=None*, *updated_at=None*, *resource_version=None*, *description=None*, *model_invariant_id=None*, *model_version_id=None*, *persona_model_version=None*, *widget_model_id=None*, *widget_model_version=None*, *bandwith_total=None*, *vhn_portal_url=None*, *service_instance_location_id=None*, *selflink=None*, *orchestration_status=None*, *input_parameters=None*)

Service instance creation.

Parameters

- **service_subscription** (ServiceSubscription) – service subscription which is belongs to

- **instance_id** (*str*) – Uniquely identifies this instance of a service
- **instance_name** (*str, optional*) – This field will store a name assigned to the service-instance. Defaults to None.
- **service_type** (*str, optional*) – String capturing type of service. Defaults to None.
- **service_role** (*str, optional*) – String capturing the service role. Defaults to None.
- **environment_context** (*str, optional*) – This field will store the environment context assigned to the service-instance. Defaults to None.
- **workload_context** (*str, optional*) – This field will store the workload context assigned to the service-instance. Defaults to None.
- **created_at** (*str, optional*) – Create time of Network Service. Defaults to None.
- **updated_at** (*str, optional*) – Last update of Network Service. Defaults to None.
- **description** (*str, optional*) – Short description for service-instance. Defaults to None.
- **model_invariant_id** (*str, optional*) – The ASDC model id for this resource or service model. Defaults to None.
- **model_version_id** (*str, optional*) – The ASDC model version for this resource or service model. Defaults to None.
- **persona_model_version** (*str, optional*) – The ASDC model version for this resource or service model. Defaults to None.
- **widget_model_id** (*str, optional*) – The ASDC data dictionary widget model. This maps directly to the A&AI widget. Defaults to None.
- **widget_model_version** (*str, optional*) – The ASDC data dictionary version of the widget model. This maps directly to the A&AI version of the widget. Defaults to None.
- **bandwith_total** (*str, optional*) – Indicates the total bandwidth to be used for this service. Defaults to None.
- **vhn_portal_url** (*str, optional*) – URL customers will use to access the vHN Portal. Defaults to None.
- **service_instance_location_id** (*str, optional*) – An identifier that customers assign to the location where this service is being used. Defaults to None.
- **resource_version** (*str, optional*) – Used for optimistic concurrency. Must be empty on create, valid on update and delete. Defaults to None.
- **selflink** (*str, optional*) – Path to the controller object. Defaults to None.
- **orchestration_status** (*str, optional*) – Orchestration status of this service. Defaults to None.
- **input_parameters** (*str, optional*) – String capturing request parameters from SO to pass to Closed Loop. Defaults to None.

delete(*a_la_carte=True*)

Create service deletion request.

Send a request to delete service instance

Parameters **a_la_carte** (*boolean*) – deletion mode

Returns Deletion request object

Return type *ServiceDeletionRequest*

classmethod get_all_url(*service_subscription*)
Return an url to get all service instances for service subscription.

Parameters **service_subscription** (ServiceSubscription) – Service subscription object

Returns Url to get all service instances for service subscription

Return type str

property network_instances
Network instances associated with service instance.

Returns iterator of NetworkInstance representing network instantiated for that service

Yields *NetworkInstance* – NetworkInstance object

Return type Iterator[*NetworkInstance*]

property pnfs
Pnfs associated with service instance.

Returns iterator of PnfInstance representing pnfs instantiated for that service

Yields *PnfInstance* – PnfInstance object

Return type Iterator[*PnfInstance*]

property sdc_service
Sdc service related with that instance.

Sdc service model which was used to create that instance.

Raises ***ResourceNotFound*** – Service model not found

Return type *Service*

property url
Service instance resource URL.

Returns Service instance url

Return type str

property vnf_instances
Vnf instances associated with service instance.

Returns iterator of VnfInstances representing VNF instantiated for that service

Yields *VnfInstance* – VnfInstance object

Return type Iterator[*VnfInstance*]

onapsdk.aai.business.sp_partner module

A&AI sp-partner module.

class onapsdk.aai.business.sp_partner.**SpPartner**(*sp_partner_id*, *resource_version*, *url=None*, *callsource=None*, *operational_status=None*, *model_customization_id=None*, *model_invariant_id=None*, *model_version_id=None*)

Bases: *onapsdk.aai.aai_element.AaiResource*

Sp partner class.

classmethod create(*sp_partner_id*, *url=''*, *callsource=''*, *operational_status=''*, *model_customization_id=''*, *model_invariant_id=''*, *model_version_id=''*)

Create sp partner A&AI resource.

Parameters

- **sp_partner_id** (*str*) – sp partner unique ID
- **url** (*str, optional*) – Store the URL of this sp-partner. Defaults to None
- **callsource** (*str, optional*) – Store the callsource of this sp-partner. Defaults to None
- **operational_status** (*str, optional*) – Store the operational-status of this sp-partner. Defaults to None
- **model_customization_id** (*str, optional*) – Store the model-customization-id of this sp-partner. Defaults to None
- **model_invariant_id** (*str, optional*) – The ASDC model id for this sp-partner model. Defaults to None
- **model_version_id** (*str, optional*) – The ASDC model version for this sp-partner model. Defaults to None

Returns Created SpPartner object

Return type *SpPartner*

classmethod get_all()

Get all sp partners.

Yields *SpPartner* – SpPartner object

Return type Iterator[*SpPartner*]

classmethod get_all_url()

Return url to get all sp partners.

Returns Url to get all sp partners

Return type str

classmethod get_by_sp_partner_id(*sp_partner_id*)

Get sp partner by it's ID.

Parameters **sp_partner_id** (*str*) – sp partner object id

Returns SpPartner object

Return type *SpPartner*

property url
Sp partner's url.

Returns Resource's url

Return type str

onapsdk.aai.business.vf_module module

VF module instance.

class onapsdk.aai.business.vf_module.**VfModuleInstance**(*vnf_instance*, *vf_module_id*, *is_base_vf_module*, *automated_assignment*, *vf_module_name=None*, *heat_stack_id=None*, *resource_version=None*, *model_invariant_id=None*, *orchestration_status=None*, *persona_model_version=None*, *model_version_id=None*, *model_customization_id=None*, *widget_model_id=None*, *widget_model_version=None*, *contrail_service_instance_fqdn=None*, *module_index=None*, *selflink=None*)

Bases: *onapsdk.aai.business.instance.Instance*

Vf module instance class.

classmethod create_from_api_response(*api_response*, *vnf_instance*)
Create vf module instance object using HTTP API response dictionary.

Parameters

- **api_response** (*dict*) – HTTP API response content
- **vnf_instance** (VnfInstance) – VnfInstance associated with VfModuleInstance

Returns VfModuleInstance object

Return type *VfModuleInstance*

delete(*a_la_carte=True*)
Create deletion request.

Send request to delete VF module instance

Parameters **a_la_carte** (*boolean*) – deletion mode

Returns Deletion request object

Return type *VfModuleDeletionRequest*

classmethod get_all_url(*vnf_instance*)
Return url to get all vf modules for vnf instance.

Parameters **vnf_instance** (VnfInstance) – VNF instance object

Returns Url to get all vf modules for vnf instance

Return type str

property url
Resource url.

Returns VfModuleInstance url

Return type str

property vf_module
Vf module associated with that vf module instance.

Returns VfModule object associated with vf module instance

Return type *VfModule*

onapsdk.aai.business.vnf module

Vnf instance module.

class onapsdk.aai.business.vnf.**VnfInstance**(*service_instance*, *vnf_id*, *vnf_type*, *in_maint*, *is_closed_loop_disabled*, *vnf_name=None*, *service_id=None*, *regional_resource_zone=None*, *prov_status=None*, *operational_status=None*, *equipment_role=None*, *orchestration_status=None*, *vnf_package_name=None*, *vnf_discriptor_name=None*, *job_id=None*, *heat_stack_id=None*, *mso_catalog_key=None*, *management_option=None*, *ipv4_oam_address=None*, *ipv4_loopback0_address=None*, *nm_lan_v6_address=None*, *management_v6_address=None*, *vcpu=None*, *vcpu_units=None*, *vmemory=None*, *vmemory_units=None*, *vdisk=None*, *vdisk_units=None*, *nshd=None*, *nvm=None*, *nnet=None*, *resource_version=None*, *encrypted_access_flag=None*, *model_invariant_id=None*, *model_version_id=None*, *persona_model_version=None*, *model_customization_id=None*, *widget_model_id=None*, *widget_model_version=None*, *as_number=None*, *regional_resource_subzone=None*, *nf_type=None*, *nf_function=None*, *nf_role=None*, *nf_naming_code=None*, *selflink=None*, *ipv4_oam_gateway_address=None*, *ipv4_oam_gateway_address_prefix_length=None*, *vlan_id_outer=None*, *nm_profile_name=None*)

Bases: *onapsdk.aai.business.instance.Instance*

VNF Instance class.

add_vf_module(*vf_module*, *cloud_region=None*, *tenant=None*, *vf_module_instance_name=None*, *vnf_parameters=None*, *use_preload=True*)

Instantiate vf module for that VNF instance.

Parameters

- **vf_module** (VfModule) – VfModule to instantiate
- **cloud_region** (CloudRegion, *optional*) – Cloud region to use in instantiation request. Defaults to None. THAT PROPERTY WILL BE REQUIRED IN ONE OF THE FUTURE RELEASE. REFACTOR YOUR CODE TO USE IT!.
- **tenant** (Tenant, *optional*) – Tenant to use in instnatiation request. Defaults to None. THAT PROPERTY WILL BE REQUIRED IN ONE OF THE FUTURE RELEASE. REFACTOR YOUR CODE TO USE IT!.
- **vf_module_instance_name** (*str*, *optional*) – VfModule instance name. Defaults to None.
- **vnf_parameters** (*Iterable[InstantiationParameter]*, *optional*) – InstantiationParameter to use for preloading or to be passed as "userParams". Defaults to None.
- **use_preload** (*bool*, *optional*) – Based on this flag InstantiationParameters are passed in preload or as "userParam" in the request. Defaults to True

Returns VfModuleInstantiation request object

Return type *VfModuleInstantiation*

classmethod create_from_api_response(*api_response*, *service_instance*)

Create vnf instance object using HTTP API response dictionary.

Returns VnfInstance object

Return type *VnfInstance*

delete(*a_la_carte=True*)

Create VNF deletion request.

Send request to delete VNF instance

Parameters **a_la_carte** (*boolean*) – deletion mode

Returns Deletion request

Return type *VnfDeletionRequest*

classmethod get_all_url()

Return url to get all vnfs.

Returns Url to get all vnfs

Return type str

healthcheck()

Execute healthcheck operation for vnf instance.

Returns VnfInstantiation object.

Return type *VnfInstantiation*

update(*vnf_parameters=None*)

Update vnf instance.

Parameters

- **vnf_parameters** (*Iterable["InstantiationParameter"], Optional*) – list of instantiation
- **for update operation.** (*parameters*) –

Raises ***StatusError*** – Skip post instantiation configuration flag for VF to True. It might cause problems with SO component.

Returns VnfInstantiation object.

Return type *VnfInstantiation*

property url

Vnf instance url.

Returns VnfInstance url

Return type str

property vf_modules

Vf modules associated with vnf instance.

Yields *VfModuleInstance* – VfModuleInstance associated with VnfInstance

Return type Iterator[*VfModuleInstance*]

property vnf

Vnf associated with that vnf instance.

Raises ***ResourceNotFound*** – Could not find VNF for that VNF instance

Returns Vnf object associated with vnf instance

Return type *Vnf*

Module contents

A&AI business package.

onapsdk.aai.cloud_infrastructure package

Submodules

onapsdk.aai.cloud_infrastructure.cloud_region module

Cloud region module.

class onapsdk.aai.cloud_infrastructure.cloud_region.**AvailabilityZone**(*name*, *hypervisor_type*, *operational_status=None*, *resource_version=None*)

Bases: object

Availability zone.

A collection of compute hosts/pservers

hypervisor_type: *str*

name: *str*

operational_status: *str* = None

resource_version: *str* = None

class onapsdk.aai.cloud_infrastructure.cloud_region.**CloudRegion**(*cloud_owner*, *cloud_region_id*, *orchestration_disabled*, *in_maint*, *, *cloud_type=''*, *owner_defined_type=''*, *cloud_region_version=''*, *identity_url=''*, *cloud_zone=''*, *complex_name=''*, *sriov_automation=''*, *cloud_extra_info=''*, *upgrade_cycle=''*, *resource_version=''*)

Bases: *onapsdk.aai.aai_element.AaiResource*

Cloud region class.

Represents A&AI cloud region object.

add_availability_zone(*availability_zone_name*, *availability_zone_hypervisor_type*, *availability_zone_operational_status=None*)

Add avaiability zone to cloud region.

Parameters

- **availability_zone_name** (*str*) – Name of the availability zone. Unique across a cloud region
- **availability_zone_hypervisor_type** (*str*) – Type of hypervisor
- **availability_zone_operational_status** (*str, optional*) – State that indicates whether the availability zone should be used. Defaults to None.

Return type None

add_esr_system_info(*esr_system_info_id*, *user_name*, *password*, *system_type*, *system_name=None*, *esr_type=None*, *vendor=None*, *version=None*, *service_url=None*, *protocol=None*, *ssl_cacert=None*, *ssl_insecure=None*, *ip_address=None*, *port=None*, *cloud_domain=None*, *default_tenant=None*, *passive=None*, *remote_path=None*, *system_status=None*, *openstack_region_id=None*, *resource_version=None*)

Add external system info to cloud region.

Parameters

- **esr_system_info_id** (*str*) – Unique ID of esr system info
- **user_name** (*str*) – username used to access external system

- **password** (*str*) – password used to access external system
- **system_type** (*str*) – it could be vim/vnfm/thirdparty-sdnc/ ems-resource/ems-performance/ems-alarm
- **system_name** (*str, optional*) – name of external system. Defaults to None.
- **esr_type** (*str, optional*) – type of external system. Defaults to None.
- **vendor** (*str, optional*) – vendor of external system. Defaults to None.
- **version** (*str, optional*) – version of external system. Defaults to None.
- **service_url** (*str, optional*) – url used to access external system. Defaults to None.
- **protocol** (*str, optional*) – protocol of third party SDNC, for example netconf/snmp. Defaults to None.
- **ssl_cacert** (*str, optional*) – ca file content if enabled ssl on auth-url. Defaults to None.
- **ssl_insecure** (*bool, optional*) – Whether to verify VIM's certificate. Defaults to True.
- **ip_address** (*str, optional*) – service IP of ftp server. Defaults to None.
- **port** (*str, optional*) – service port of ftp server. Defaults to None.
- **cloud_domain** (*str, optional*) – domain info for authentication. Defaults to None.
- **default_tenant** (*str, optional*) – default tenant of VIM. Defaults to None.
- **passive** (*bool, optional*) – ftp passive mode or not. Defaults to False.
- **remote_path** (*str, optional*) – resource or performance data file path. Defaults to None.
- **system_status** (*str, optional*) – he status of external system. Defaults to None.
- **openstack_region_id** (*str, optional*) – OpenStack region ID used by Multi-Cloud plugin to interact with an OpenStack instance. Defaults to None.

Return type None

add_tenant(*tenant_id*, *tenant_name*, *tenant_context=None*)

Add tenant to cloud region.

Parameters

- **tenant_id** (*str*) – Unique id relative to the cloud-region.
- **tenant_name** (*str*) – Readable name of tenant
- **tenant_context** (*str, optional*) – This field will store the tenant context.. Defaults to None.

Return type None

property availability_zones

Cloud region availability zones.

Iterate over CloudRegion availability zones. Relationship list is given using A&AI API call.

Returns CloudRegion availability zone

Return type Iterator[*AvailabilityZone*]

property complex
Complex related with cloud region.

Returns

Complex object related with CloudRegion or None if CloudRegion has no relationship with any Complex

Return type Optional[*Complex*]

classmethod create(*cloud_owner*, *cloud_region_id*, *orchestration_disabled*, *in_maint*, ***, *cloud_type=''*, *owner_defined_type=''*, *cloud_region_version=''*, *identity_url=''*, *cloud_zone=''*, *complex_name=''*, *sriov_automation=''*, *cloud_extra_info=''*, *upgrade_cycle=''*)
Create CloudRegion object.

Create cloud region with given values.

Returns Created cloud region.

Return type *CloudRegion*

delete()
Delete cloud region.

Return type None

property esr_system_infos
Cloud region collection of persistent block-level external system auth info.

Returns Cloud region external system address information.

Return type Iterator[*EsrSystemInfo*]

classmethod get_all(*cloud_owner=None*, *cloud_region_id=None*, *cloud_type=None*, *owner_defined_type=None*)
Get all A&AI cloud regions.

Cloud regions can be filtered by 4 parameters: cloud-owner, cloud-region-id, cloud-type and owner-defined-type.

Yields

CloudRegion – CloudRegion object. Can not yield anything if cloud region with given filter parameters doesn't exist

Return type Iterator[*CloudRegion*]

classmethod get_all_url()
Return url to get all cloud regions.

Returns Url to get all cloud regions

Return type str

get_availability_zone_by_name(*zone_name*)
Get availability zone with provided Name.

Parameters **name** (*availability_zone*) – The name of the availibilty zone

Returns AvailabilityZone object

Return type *AvailabilityZone*

classmethod get_by_id(*cloud_owner*, *cloud_region_id*)
Get CloudRegion object by cloud_owner and cloud-region-id field value.

This method calls A&AI cloud region API filtering them by cloud_owner and cloud-region-id field value.

Raises ***ResourceNotFound*** – Cloud region with given id does not exist.

Returns CloudRegion object with given cloud-region-id.

Return type *CloudRegion*

get_tenant(*tenant_id*)
Get tenant with provided ID.

Parameters **tenant_id** (*str*) – Tenant ID

Returns Tenant object

Return type *Tenant*

get_tenants_by_name(*tenant_name*)
Get tenants with given name.

Parameters **tenant_name** (*str*) – Tenant name

Returns Iterate through cloud region tenants with given name

Return type Iterator[*Tenant*]

link_to_complex(*complex_object*)
Link cloud region to comples.

It creates relationhip object and add it into cloud region.

Return type None

register_to_multicloud(*default_tenant=None*)
Register cloud to multicloud using MSB API.

Parameters **default_tenant** (*str, optional*) – Default tenant. Defaults to None.

Return type None

property tenants
Tenants iterator.

Cloud region tenants iterator.

Returns Iterate through cloud region tenants

Return type Iterator[*Tenant*]

unregister_from_multicloud()
Unregister cloud from mutlicloud.

Return type None

property url
Cloud region object url.

URL used to call CloudRegion A&AI API

Returns CloudRegion object url

Return type str

class onapsdk.aai.cloud_infrastructure.cloud_region.**EsrSystemInfo**(*esr_system_info_id*, *user_name*, *password*, *system_type*, *resource_version*, *system_name=None*, *esr_type=None*, *vendor=None*, *version=None*, *service_url=None*, *protocol=None*, *ssl_cacert=None*, *ssl_insecure=None*, *ip_address=None*, *port=None*, *cloud_domain=None*, *default_tenant=None*, *passive=None*, *remote_path=None*, *system_status=None*, *openstack_region_id=None*)

Bases: object

Persist common address information of external systems.

cloud_domain: str = None

default_tenant: str = None

esr_system_info_id: str

esr_type: str = None

ip_address: str = None

openstack_region_id: str = None

passive: Optional[bool] = None

password: str

port: str = None

protocol: str = None

remote_path: str = None

resource_version: str

service_url: str = None

ssl_cacert: str = None

ssl_insecure: Optional[bool] = None

system_name: str = None

system_status: str = None

system_type: str

user_name: str

vendor: str = None

version: str = None

onapsdk.aai.cloud_infrastructure.complex module

A&AI Complex module.

class onapsdk.aai.cloud_infrastructure.complex.**Complex**(*physical_location_id*, ***, *name=''*, *data_center_code=''*, *identity_url=''*, *resource_version=''*, *physical_location_type=''*, *street1=''*, *street2=''*, *city=''*, *state=''*, *postal_code=''*, *country=''*, *region=''*, *latitude=''*, *longitude=''*, *elevation=''*, *lata=''*, *timezone=''*, *data_owner=''*, *data_source=''*, *data_source_version=''*)

Bases: *onapsdk.aai.aai_element.AaiResource*

Complex class.

Collection of physical locations that can house cloud-regions.

classmethod create(*physical_location_id*, ***, *name=''*, *data_center_code=''*, *identity_url=''*, *resource_version=''*, *physical_location_type=''*, *street1=''*, *street2=''*, *city=''*, *state=''*, *postal_code=''*, *country=''*, *region=''*, *latitude=''*, *longitude=''*, *elevation=''*, *lata=''*, *timezone=''*, *data_owner=''*, *data_source=''*, *data_source_version=''*)

Create complex.

Create complex object by calling A&AI API. If API request doesn't fail it returns Complex object.

Returns Created complex object

Return type *Complex*

classmethod create_from_api_response(*api_response*)

Create complex object using given A&AI API response JSON.

Parameters **api_response** (*Dict[str, Any]*) – Complex A&AI API response

Returns Complex object created from given response

Return type *Complex*

delete()
Delete complex.

Return type None

classmethod get_all(*physical_location_id=None*, *data_center_code=None*, *complex_name=None*, *identity_url=None*)
Get all complexes from A&AI.

Call A&AI API to get all complex objects.

Parameters

- **physical_location_id** (*str, optional*) – Unique identifier for physical location, e.g., CLLI. Defaults to None.
- **data_center_code** (*str, optional*) – Data center code which can be an alternate way to identify a complex. Defaults to None.
- **complex_name** (*str, optional*) – Gamma complex name for LCP instance. Defaults to None.
- **identity_url** (*str, optional*) – URL of the keystone identity service. Defaults to None.

Yields

Complex – Complex object. Can not yield anything if any complex with given filter parameters doesn't exist

Return type Iterator[*Complex*]

classmethod get_all_url()
Return an url to get all complexes.

Returns URL to get all complexes

Return type str

classmethod get_by_physical_location_id(*physical_location_id*)
Get complex by physical location id.

Parameters **physical_location_id** (*str*) – Physical location id of Complex

Returns Complex object

Return type *Complex*

Raises ***ResourceNotFound*** – Complex with given physical location id not found

property url
Complex url.

Returns Complex url

Return type str

onapsdk.aai.cloud_infrastructure.tenant module

A&AI Tenant module.

class onapsdk.aai.cloud_infrastructure.tenant.**Tenant**(*cloud_region*, *tenant_id*, *tenant_name*, *tenant_context=None*, *resource_version=None*)

Bases: *onapsdk.aai.aai_element.AaiResource*

Tenant class.

delete()
Delete tenant.

Remove tenant from cloud region.

Return type None

classmethod get_all_url(*cloud_region*)
Return an url to get all tenants for given cloud region.

Parameters **cloud_region** (CloudRegion) – Cloud region object

Returns Url to get all tenants

Return type str

property url
Tenant url.

Returns Url which can be used to update or delete tenant.

Return type str

Module contents

A&AI cloud infrastructure package.

Submodules

onapsdk.aai.aai_element module

AAI Element module.

class onapsdk.aai.aai_element.**AaiElement**
Bases: *onapsdk.onap_service.OnapService*

Mother Class of all A&AI elements.

api_version = '/aai/v23'

base_url = 'https://aai.api.sparky.simpledemo.onap.org:30233'

classmethod get_guis()
Retrieve the status of the AAI GUIs.

Only one GUI is referenced for AAI the AAI sparky GUI

Return the list of GUIs

Return type GuiItem

headers: Dict[str, str] = {'Accept': 'application/json', 'Content-Type': 'applicati

name: str = 'AAI'

server: str = 'AAI'

class onapsdk.aai.aai_element.**AaiResource**

Bases: *onapsdk.aai.aai_element.AaiElement*

A&AI resource class.

add_relationship(*relationship*)

Add relationship to aai resource.

Add relationship to resource using A&AI API

Parameters **relationship** (Relationship) – Relationship to add

Return type None

classmethod count(**args*, ***kwargs*)

Get the count number of all objects of given class.

Get the response, iterate through response (each class has different response) – the first key value is the count.

Returns Count of the objects

Return type int

classmethod filter_none_key_values(*dict_to_filter*)

Filter out None key values from dictionary.

Iterate through given dictionary and filter None values.

Parameters **dict_to_filter** (*Dict*) – Dictionary to filter out None

Returns:dataclasse init a field Dict[str, str]: Filtered dictionary

Return type Dict[str, str]

classmethod get_all_url(**args*, ***kwargs*)

Return an url for all objects of given class.

Returns URL to get all objects of given class

Return type str

property relationships

Resource relationships iterator.

Yields *Relationship* – resource relationship

Raises ***RelationshipNotFound*** – if request for relationships returned 404

Return type Iterator[*Relationship*]

property url

Resource's url.

Returns Resource's url

Return type str

class onapsdk.aai.aai_element.**Relationship**(*related_to*, *related_link*, *relationship_data*, *relationship_label=''*, *related_to_property=<factory>*)

Bases: object

Relationship class.

A&AI elements could have relationship with other A&AI elements. Relationships are represented by this class objects.

get_relationship_data(*relationship_key*)

Get relationship data for given relationship key.

From list of relationship data get the value for given key

Parameters **relationship_key** (*str*) – Key to get relationship data value

Returns

Relationship value or None if relationship data with provided ket doesn't exist

Return type Optional[str]

related_link: **str**

related_to: **str**

related_to_property: **List[Dict[str, str]]**

relationship_data: **List[Dict[str, str]]**

relationship_label: **str** = ''

onapsdk.aai.bulk module

onapsdk.aai.service_design_and_creation module

AAI service-design-and-creation module.

class onapsdk.aai.service_design_and_creation.**Model**(*invariant_id*, *model_type*, *resource_version*)

Bases: *onapsdk.aai.aai_element.AaiResource*

Model resource class.

classmethod get_all()

Get all models.

Yields *Model* – Model object

Return type Iterator[*Model*]

classmethod get_all_url()

Return url to get all models.

Returns Url to get all models

Return type str

property url

Model instance url.

Returns Model's url

Return type str

class onapsdk.aai.service_design_and_creation.**Service**(*service_id*, *service_description*, *resource_version*)

Bases: *onapsdk.aai.aai_element.AaiResource*

SDC service class.

classmethod create(*service_id*, *service_description*)

Create service.

Parameters

- **service_id** (*str*) – service ID
- **service_description** (*str*) – service description

Return type None

classmethod get_all(*service_id=None*, *service_description=None*)

Services iterator.

Stand-in for service model definitions.

Returns Service

Return type Iterator[*Service*]

classmethod get_all_url()

Return url to get all services.

Returns Url to get all services

Return type str

property url

Service object url.

Returns Service object url address

Return type str

Module contents

ONAP SDK AAI package.

onapsdk.cds package

Submodules

onapsdk.cds.blueprint module

CDS Blueprint module.

class onapsdk.cds.blueprint.**Blueprint**(*cba_file_bytes*)

Bases: *onapsdk.cds.cds_element.CdsElement*

CDS blueprint representation.

TEMPLATES_RE = 'Templates\\/.*json$'

TOSCA_META = 'TOSCA-Metadata/TOSCA.meta'

deploy()
Deploy blueprint.

Return type None

enrich()
Call CDS API to get enriched blueprint file.

Returns Enriched blueprint object

Return type *Blueprint*

static get_cba_metadata(*cba_tosca_meta_bytes*)
Parse CBA TOSCA.meta file and get values from it.

Parameters **cba_tosca_meta_bytes** (*bytes*) – TOSCA.meta file bytes.

Raises ***ValidationError*** – TOSCA Meta file has invalid format.

Returns Dataclass with CBA metadata

Return type *CbaMetadata*

get_data_dictionaries()
Get the generated data dictionaries required by blueprint.

If mapping reqires other source than input it should be updated before upload to CDS.

Returns DataDictionary objects.

Return type Generator[*DataDictionary*, None, None]

get_mappings(*cba_zip_file*)
Read mappings from CBA file.

Search mappings in CBA file and create Mapping object for each of them.

Parameters **cba_zip_file** (*ZipFile*) – CBA file to get mappings from.

Returns Mappings set object.

Return type *MappingSet*

static get_mappings_from_mapping_file(*cba_mapping_file_bytes*)
Read mapping file and create Mappping object for it.

Parameters **cba_mapping_file_bytes** (*bytes*) – CBA mapping file bytes.

Yields *Generator[Mapping, None, None]* – Mapping object.

Return type Generator[*Mapping*, None, None]

get_resolved_template(*artifact_name*, *resolution_key=None*, *resource_type=None*, *resource_id=None*, *occurrence=None*)
Get resolved template for Blueprint.

Parameters

- **artifact_name** (*str*) – Resolved template's artifact name
- **resolution_key** (*Optional[str], optional*) – Resolved template's resolution key. Defaults to None.
- **resource_type** (*Optional[str], optional*) – Resolved template's resource type. Defaults to None.

- **resource_id** (*Optional[str], optional*) – Resolved template's resource ID. Defaults to None.
- **occurrence** (Optional[str]) – (Optional[str], optional): Resolved template's occurrence value. Defaults to None.

Returns Resolved template

Return type Dict[str, str]

get_workflow_by_name(*workflow_name*)
Get workflow by name.

If there is no workflow with given name *ParameterError* is going to be raised.

Parameters **workflow_name** (*str*) – Name of the workflow

Returns Workflow with given name

Return type *Workflow*

get_workflows(*cba_entry_definitions_file_bytes*)
Get worfklows from entry_definitions file.

Parse entry_definitions file and create Workflow objects for workflows stored in.

Parameters **cba_entry_definitions_file_bytes** (*bytes*) – entry_definition file.

Yields *Generator[Workflow, None, None]* – Workflow object.

Return type Generator[*Workflow*, None, None]

classmethod load_from_file(*cba_file_path*)
Load blueprint from file.

Raises ***FileError*** – File to load blueprint from doesn't exist.

Returns Blueprint object

Return type *Blueprint*

property mappings
Blueprint mappings collection.

Returns Mappings collection.

Return type *MappingSet*

property metadata
Blueprint metadata.

Data from TOSCA.meta file.

Returns Blueprint metadata object.

Return type *CbaMetadata*

publish()
Publish blueprint.

Return type None

save(*dest_file_path*)
Save blueprint to file.

Parameters **dest_file_path** (*str*) – Path of file where blueprint is going to be saved

Return type None

store_resolved_template(*artifact_name*, *data*, *resolution_key=None*, *resource_type=None*, *resource_id=None*)

Store resolved template for Blueprint.

Parameters

- **artifact_name** (*str*) – Resolved template's artifact name
- **data** (*str*) – Resolved template
- **resolution_key** (*Optional[str], optional*) – Resolved template's resolution key. Defaults to None.
- **resource_type** (*Optional[str], optional*) – Resolved template's resource type. Defaults to None.
- **resource_id** (*Optional[str], optional*) – Resolved template's resource ID. Defaults to None.

Return type None

property url

URL address to use for CDS API call.

Returns URL to CDS blueprintprocessor.

Return type str

property workflows

Blueprint's workflows property.

Returns Blueprint's workflow list.

Return type List[*Workflow*]

class onapsdk.cds.blueprint.**CbaMetadata**(*tosca_meta_file_version*, *csar_version*, *created_by*, *entry_definitions*, *template_name*, *template_version*, *template_tags*)

Bases: object

Class to hold CBA metadata values.

created_by: **str**

csar_version: **str**

entry_definitions: **str**

template_name: **str**

template_tags: **str**

template_version: **str**

tosca_meta_file_version: **str**

class onapsdk.cds.blueprint.**Mapping**(*name*, *mapping_type*, *dictionary_name*, *dictionary_sources=<factory>*)

Bases: object

Blueprint's template mapping.

Stores mapping data:

- name,
- type,
- name of dictionary from which value should be get,

- dictionary source of value.

dictionary_name: str

dictionary_sources: List[str]

generate_data_dictionary()
Generate data dictionary for mapping.

Data dictionary with required data sources, type and name for mapping will be created from Jinja2 template.

Returns Data dictionary

Return type dict

mapping_type: str

merge(*mapping*)
Merge mapping objects.

Merge objects dictionary sources.

Parameters **mapping** (Mapping) – Mapping object to merge.

Return type None

name: str

class onapsdk.cds.blueprint.**MappingSet**
Bases: object

Set of mapping objects.

Mapping objects will be stored in dictionary where mapping name is a key. No two mappings with the same name can be stored in this collection.

add(*mapping*)
Add mapping to set.

If there is already mapping object with the same name in collection they will be merged.

Parameters **mapping** (Mapping) – Mapping to add to collection.

Return type None

extend(*iterable*)
Extend set with an iterator of mappings.

Parameters **iterable** (*Iterator[Mapping]*) – Mappings iterator.

Return type None

class onapsdk.cds.blueprint.**ResolvedTemplate**(*blueprint*, *artifact_name=None*, *resolution_key=None*, *resource_id=None*, *resource_type=None*, *occurrence=None*, *response_format='application/json'*)
Bases: *onapsdk.cds.cds_element.CdsElement*

Resolved template class.

Store and retrieve rendered template results.

get_resolved_template()
Get resolved template.

Returns Resolved template

Return type Dict[str, str]

property resolved_template_url
Url to retrieve resolved template.

Filter None parameters.

Returns Retrieve resolved template url

Return type str

store_resolved_template(*resolved_template*)
Store resolved template.

Parameters **resolved_template** (*str*) – Template to store

Raises

- ***ParameterError*** – To store template it's needed to pass artifact name and: - resolution key, or - resource type and resource id.
- **If not all needed parameters are given that exception will be raised.** –

Return type None

store_resolved_template_with_resolution_key(*resolved_template*)
Store template using resolution key.

Parameters **resolved_template** (*str*) – Template to store

Return type None

store_resolved_template_with_resource_type_and_id(*resolved_template*)
Store template using resource type and resource ID.

Parameters **resolved_template** (*str*) – Template to store

Return type None

property url
Url property.

Returns Url

Return type str

class onapsdk.cds.blueprint.**Workflow**(*cba_workflow_name*, *cba_workflow_data*, *blueprint*)
Bases: *onapsdk.cds.cds_element.CdsElement*

Blueprint's workflow.

Stores workflow steps, inputs, outputs. Executes workflow using CDS HTTP API.

class WorkflowInput(*name*, *required*, *type*, *description=''*)
Bases: object

Workflow input class.

Stores input name, information if it's required, type, and optional description.

description: str = ''

name: str

required: bool

type: str

class WorkflowOutput(*name*, *type*, *value*)
Bases: object

Workflow output class.

Stores output name, type na value.

name: str

type: str

value: Dict[str, Any]

class WorkflowStep(*name*, *description*, *target*, *activities=<factory>*)
Bases: object

Workflow step class.

Stores step name, description, target and optional activities.

activities: List[Dict[str, str]]

description: str

name: str

target: str

execute(*inputs*)
Execute workflow.

Call CDS HTTP API to execute workflow.

Parameters **inputs** (*dict*) – Inputs dictionary.

Returns Response's payload.

Return type dict

property inputs
Workflow's inputs property.

Returns List of workflows's inputs.

Return type List[*Workflow.WorkflowInput*]

property outputs
Workflow's outputs property.

Returns List of workflows's outputs.

Return type List[*Workflow.WorkflowOutput*]

property steps
Workflow's steps property.

Returns List of workflow's steps.

Return type List[*Workflow.WorkflowStep*]

property url
Workflow execution url.

Returns Url to call warkflow execution.

Return type str

onapsdk.cds.blueprint_model module

CDS Blueprint Models module.

class onapsdk.cds.blueprint_model.**BlueprintModel**(*blueprint_model_id*, *artifact_uuid=None*, *artifact_type=None*, *artifact_version=None*, *artifact_description=None*, *internal_version=None*, *created_date=None*, *artifact_name=None*, *published='N'*, *updated_by=None*, *tags=None*)

Bases: *onapsdk.cds.cds_element.CdsElement*

Blueprint Model class.

Represents blueprint models in CDS

delete()
Delete blueprint model.

classmethod get_all()
Get all blueprint models.

Yields *BlueprintModel* – BlueprintModel object.

Return type Iterator[*BlueprintModel*]

get_blueprint()
Get Blueprint object for selected blueprint model.

Returns Blueprint object

Return type *Blueprint*

classmethod get_by_id(*blueprint_model_id*)
Retrieve blueprint model with provided ID.

Args: blueprint_model_id (str):

Returns Blueprint model object

Return type *BlueprintModel*

Raises ***ResourceNotFound*** – Blueprint model with provided ID doesn't exist

classmethod get_by_name_and_version(*blueprint_name*, *blueprint_version*)
Retrieve blueprint model with provided name and version.

Parameters

- **blueprint_name** (*str*) – Blueprint model name
- **blueprint_version** (*str*) – Blueprint model version

Returns Blueprint model object

Return type *BlueprintModel*

Raises ***ResourceNotFound*** – Blueprint model with provided name and version doesn't exist

save(*dst_file_path*)
Save blueprint model to file.

Parameters **dst_file_path** (*str*) – Path of file where blueprint is going to be saved

onapsdk.cds.blueprint_processor module

CDS Blueprintprocessor module.

class onapsdk.cds.blueprint_processor.**Blueprintprocessor**
Bases: *onapsdk.cds.cds_element.CdsElement*

Blueprintprocessor class.

classmethod bootstrap(*load_model_type=True*, *load_resource_dictionary=True*, *load_cba=True*)
Bootstrap CDS blueprintprocessor.

That action in needed to work with CDS. Can be done only once.

Parameters

- **load_model_type** (*bool, optional*) – Datermines if model types should be loaded on bootstrap. Defaults to True.
- **load_resource_dictionary** (*bool, optional*) – Determines if resource dictionaries should be loaded on bootstrap. Defaults to True.
- **load_cba** (*bool, optional*) – Determines if cba files should be loaded on bootstrap. Defaults to True.

Return type None

onapsdk.cds.cds_element module

Base CDS module.

class onapsdk.cds.cds_element.**CdsElement**
Bases: *onapsdk.onap_service.OnapService*, abc.ABC

Base CDS class.

Stores url to CDS API (edit if you want to use other) and authentication tuple (username, password).

auth: tuple = ('ccsdkapps', 'ccsdkapps')

classmethod get_guis()
Retrieve the status of the CDS GUIs.

Only one GUI is referenced for CDS: CDS UI

Return the list of GUIs

Return type GuiItem

onapsdk.cds.data_dictionary module

CDS data dictionary module.

class onapsdk.cds.data_dictionary.**DataDictionary**(*data_dictionary_json*, *fix_schema=True*)
Bases: *onapsdk.cds.cds_element.CdsElement*

Data dictionary class.

fix_schema()
Fix data dictionary schema.

"Raw" data dictionary can be passed during initialization, but this kind of data dictionary can't be uploaded to blueprintprocessor. That method tries to fix it. It can be done only if "raw" data dictionary has a given schema:

{ "name": "string", "tags": "string", "updated-by": "string", "property": {

"description": "string", "type": "string"

}

}

Raises ***ValidationError*** – Data dictionary doesn't have all required keys

Return type None

classmethod get_by_name(*name*)

Get data dictionary by the provided name.

Returns Data dicionary object with the given name

Return type *DataDictionary*

has_valid_schema()

Check data dictionary json schema.

Check data dictionary JSON and return bool if schema is valid or not.

Valid schema means that data dictionary has given keys:

- "name"
- "tags"
- "data_type"
- "description"
- "entry_schema"
- "updatedBy"
- "definition"

"definition" key value should contains the "raw" data dictionary.

Returns True if data dictionary has valid schema, False otherwise

Return type bool

logger: logging.Logger = <Logger onapsdk.cds.data_dictionary (WARNING)>

property name

Data dictionary name.

Returns Data dictionary name

Return type str

upload()

Upload data dictionary using CDS API.

Return type None

property url

URL to call.

Returns CDS dictionary API url

Return type str

class onapsdk.cds.data_dictionary.**DataDictionarySet**
Bases: object

Data dictionary set.

Stores data dictionary and upload to server.

add(*data_dictionary*)
Add data dictionary object to set.

Based on name it won't add duplications.

Parameters **data_dictionary** (DataDictionary) – object to add to set.

Return type None

property length
Get the length of data dicitonary set.

Returns Number of data dictionaries in set

Return type int

classmethod load_from_file(*dd_file_path*, *fix_schema=True*)
Create data dictionary set from file.

File has to have valid JSON with data dictionaries list.

Parameters

- **dd_file_path** (*str*) – Data dictionaries file path.
- **fix_schema** (*bool*) – Determines if schema should be fixed or not.

Raises ***FileError*** – File to load data dictionaries from doesn't exist.

Returns Data dictionary set with data dictionaries from given file.

Return type *DataDictionarySet*

logger: logging.Logger = <Logger onapsdk.cds.data_dictionary (WARNING)>

save_to_file(*dd_file_path*)
Save data dictionaries to file.

Parameters **dd_file_path** (*str*) – Data dictinary file path.

Return type None

upload()
Upload all data dictionaries using CDS API.

Raises **RuntimeError** – Raises if any data dictionary won't be uploaded to server. Data dictionaries uploaded before the one which raises excepion won't be deleted from server.

Return type None

Module contents

ONAP SDK CDS package.

onapsdk.clamp package

Submodules

onapsdk.clamp.clamp_element module

Clamp module.

class onapsdk.clamp.clamp_element.**Clamp**
Bases: *onapsdk.onap_service.OnapService*

Mother Class of all CLAMP elements.

classmethod base_url()
Give back the base url of Clamp.

Return type str

classmethod check_loop_template(*service*)
Return loop template name if exists.

Parameters **service** (Service) – the distributed sdc service with tca blueprint artifact

Raises ***ResourceNotFound*** – Template not found.

Return type str

Returns if required template exists in CLAMP or not

classmethod check_policies(*policy_name*, *req_policies=30*)
Ensure that a policy is stored in CLAMP.

Parameters

- **policy_name** (*str*) – policy acronym
- **req_policies** (*int*) – number of required policies in CLAMP

Return type bool

Returns if required policy exists in CLAMP or not

headers: Dict[str, str] = {'Accept': 'application/json', 'Authorization': 'Basic ZG

name: str = 'CLAMP'

onapsdk.clamp.loop_instance module

Control Loop module.

class onapsdk.clamp.loop_instance.**LoopInstance**(*template*, *name*, *details*)
Bases: *onapsdk.clamp.clamp_element.Clamp*

Control Loop instantiation class.

act_on_loop_policy(*func*)
Act on loop's policy.

Parameters **func** (*function*) – function of action to be done (submit, stop, restart)

Returns failed or done

Return type action state

add_drools_conf()
Add drools configuration.

Return type dict

add_frequency_limiter(*limit=1*)
Add frequency limiter config.

Return type None

add_minmax_config()
Add MinMax operational policy config.

Return type None

add_op_policy_config(*func*, ***kwargs*)
Add operational policy configuration.

Add operation policy configuration using payload data.

Parameters **func** (*function*) – policy configuration function in (add_drools_conf, add_minmax_config, add_frequency_limiter)

Return type None

add_operational_policy(*policy_type*, *policy_version*)
Add operational policy to the loop instance.

Parameters

- **policy_type** (*str*) – the full policy model type
- **policy_version** (*str*) – policy version

Raises ***ParameterError*** – Corrupt response or a key in a dictionary not found. It will also be raised when the response contains more operational policies than there are currently.

Return type None

create()
Create instance and load loop details.

Return type None

delete()
Delete the loop instance.

Return type None

deploy_microservice_to_dcae()
Execute the deploy operation on the loop instance.

Return type bool

Returns loop deploy on DCAE status (True, False)

property details
Return and lazy load the details.

Return type dict

extract_operational_policy_name(*policy_type*)
Return operational policy name for a closed loop and a given policy.

Parameters **policy_type** (*str*) – the policy acronym.

Raises ***ParameterError*** – Couldn't load the operational policy name.

Return type str

Returns Operational policy name in the loop details after adding a policy.

property loop_schema
Return and lazy load the details schema.

Return type dict

Returns schema to be respected to accede to loop details

operational_policies = ''

refresh_status()
Reshresh loop status.

Return type None

remove_operational_policy(*policy_type*, *policy_version*)
Remove operational policy from the loop instance.

Parameters

- **policy_type** (*str*) – the full policy model type
- **policy_version** (*str*) – policy version

Return type None

restart()
Redeploy policies to policy engine.

stop()
Undeploy Policies from policy engine.

submit()
Submit policies to policy engine.

undeploy_microservice_from_dcae()
Stop the deploy operation.

Return type None

update_microservice_policy()
Update microservice policy configuration.

Update microservice policy configuration using payload data.

Return type None

validate_details()
Validate Loop Instance details.

Return type bool

Returns schema validation status (True, False)

Module contents

ONAP SDK CLAMP package.

onapsdk.configuration package

Submodules

onapsdk.configuration.global_settings module

Global settings module.

onapsdk.configuration.loader module

Settings loader module.

class onapsdk.configuration.loader.**SettingsLoader**
Bases: object

Settings loader class.

Load global settings and optionally load custom settings by importing the module stored in ONAP_PYTHON_SDK_SETTINGS environment variable. The module has to be uder PYTHONPATH.

filter_and_set(*module*)
Filter module attributes and save the uppercased.

Iterate through module's attribures and save the value of them which name is uppercase.

Parameters **module** (*module*) – Module to get attributes from

Return type None

Module contents

Configuration module.

onapsdk.dmaap package

Submodules

onapsdk.dmaap.dmaap module

Base Dmaap event store.

class onapsdk.dmaap.dmaap.**Dmaap**
Bases: *onapsdk.dmaap.dmaap_service.DmaapService*

Dmaap library provides functions for getting events from Dmaap.

dmaap_url = 'http://dmaap.api.simpledemo.onap.org:3904'

classmethod get_all_events(*basic_auth*)
Get all events stored in Dmaap.

Parameters **basic_auth** (Dict[str, str]) – (Dict[str, str]) for example:{ 'username': 'bob', 'password': 'secret' }

Return type dict

Returns (dict) Events from Dmaap

get_all_events_url = 'http://dmaap.api.simpledemo.onap.org:3904/events'

classmethod get_all_topics(*basic_auth*)
Get all topics stored in Dmaap.

Parameters **basic_auth** (Dict[str, str]) – (Dict[str, str]) for example:{ 'username': 'bob', 'password': 'secret' }

Return type dict

Returns (dict) Topics from Dmaap

get_all_topics_url = 'http://dmaap.api.simpledemo.onap.org:3904/topics'

classmethod get_events_for_topic(*topic*, *basic_auth*)
Get all events stored specific topic in Dmaap.

Parameters

- **topic** (str) – (str) topic of events stored in Dmaap
- **basic_auth** (Dict[str, str]) – (Dict[str, str]) for example:{ 'username': 'bob', 'password': 'secret' }

Return type dict

Returns (dict) Events from Dmaap

get_events_from_topic_url = '{}/events/{}/CG1/C1'

onapsdk.dmaap.dmaap_service module

Base VES module.

class onapsdk.dmaap.dmaap_service.**DmaapService**
Bases: *onapsdk.onap_service.OnapService*

Base DMAAP class.

Stores url to DMAAP API (edit if you want to use other).

Module contents

ONAP SDK Dmaap package.

onapsdk.msb package

Subpackages

onapsdk.msb.k8s package

Submodules

onapsdk.msb.k8s.connectivity_info module

Connectivity-Info module.

class onapsdk.msb.k8s.connectivity_info.**ConnectivityInfo**(*cloud_region_id*, *cloud_owner*, *other_connectivity_list*, *kubeconfig*)

Bases: *onapsdk.msb.msb_service.MSB*

Connectivity-Info class.

api_version = '/api/multicloud-k8s/v1/v1'

classmethod create(*cloud_region_id*, *cloud_owner*, *kubeconfig=None*)

Create Connectivity Info.

Parameters

- **cloud_region_id** (*str*) – Cloud region ID
- **cloud_owner** (*str*) – Cloud owner name
- **kubeconfig** (*bytes*) – kubernetes cluster kubeconfig file

Returns Created object

Return type *ConnectivityInfo*

delete()

Delete connectivity info.

Return type None

classmethod get_connectivity_info_by_region_id(*cloud_region_id*)

Get connectivity-info by its name (cloud region id).

Parameters **cloud_region_id** (*str*) – Cloud region ID

Returns Connectivity-Info object

Return type *ConnectivityInfo*

url = 'https://msb.api.simpledemo.onap.org:30283/api/multicloud-k8s/v1/v1/connectivity

onapsdk.msb.k8s.definition module

Definition module.

class onapsdk.msb.k8s.definition.**ConfigurationTemplate**(*rb_name*, *rb_version*, *template_name*, *description=''*)

Bases: *onapsdk.msb.k8s.definition.DefinitionBase*

ConfigurationTemplate class.

property url

URL address for ConfigurationTemplate calls.

Returns URL to Configuration template in Multicloud-k8s API.

Return type str

class onapsdk.msb.k8s.definition.**Definition**(*rb_name*, *rb_version*, *chart_name*, *description*, *labels*)

Bases: *onapsdk.msb.k8s.definition.DefinitionBase*

Definition class.

classmethod create(*rb_name*, *rb_version*, *chart_name=''*, *description=''*, *labels=None*)

Create Definition.

Parameters

- **rb_name** (*str*) – Definition name
- **rb_version** (*str*) – Definition version
- **chart_name** (*str*) – Chart name, optional field, will be detected if it is not provided
- **description** (*str*) – Definition description
- **labels** (*str*) – Labels

Returns Created object

Return type *Definition*

create_configuration_template(*template_name*, *description=''*)

Create configuration template.

Parameters

- **template_name** (*str*) – Name of the template
- **description** (*str*) – Description

Returns Created object

Return type *ConfigurationTemplate*

create_profile(*profile_name*, *namespace*, *kubernetes_version*, *release_name=None*)

Create Profile for Definition.

Parameters

- **profile_name** (*str*) – Name of profile
- **namespace** (*str*) – Namespace that service is created in
- **kubernetes_version** (*str*) – Required Kubernetes version
- **release_name** (*str*) – Release name

Returns Created object

Return type *Profile*

classmethod get_all()
Get all definitions.

Yields *Definition* – Definition object

get_all_configuration_templates()
Get all configuration templates.

Yields *ConfigurationTemplate* – ConfigurationTemplate object

get_all_profiles()
Get all profiles.

Yields *Profile* – Profile object

Return type Iterator[*Profile*]

get_configuration_template_by_name(*template_name*)
Get configuration template.

Parameters **template_name** (*str*) – Name of the template

Returns object

Return type *ConfigurationTemplate*

classmethod get_definition_by_name_version(*rb_name*, *rb_version*)
Get definition by it's name and version.

Parameters

- **rb_name** (*str*) – definition name
- **rb_version** (*str*) – definition version

Returns Definition object

Return type *Definition*

get_profile_by_name(*profile_name*)
Get profile by it's name.

Parameters **profile_name** (*str*) – profile name

Returns Profile object

Return type *Profile*

class onapsdk.msb.k8s.definition.**DefinitionBase**(*rb_name*, *rb_version*)
Bases: *onapsdk.msb.msb_service.MSB*

DefinitionBase class.

base_url = 'https://msb.api.simpledemo.onap.org:30283/api/multicloud-k8s/v1/v1/rb/defi

delete()
Delete Definition Based object.

Return type None

upload_artifact(*package=None*)
Upload artifact.

Parameters **package** (*bytes*) – Artifact to be uploaded to multicloud-k8s plugin

property url
URL address for Definition Based calls.

Returns URL to RB Definition

Return type str

class onapsdk.msb.k8s.definition.**Profile**(*rb_name*, *rb_version*, *profile_name*, *namespace*, *kubernetes_version*, *labels=None*, *release_name=None*)
Bases: *onapsdk.msb.k8s.definition.ProfileBase*

Profile class.

class onapsdk.msb.k8s.definition.**ProfileBase**(*rb_name*, *rb_version*, *profile_name*)
Bases: *onapsdk.msb.k8s.definition.DefinitionBase*

ProfileBase class.

property url
URL address for Profile calls.

Returns URL to RB Profile

Return type str

onapsdk.msb.k8s.instance module

Instantiation module.

class onapsdk.msb.k8s.instance.**Instance**(*instance_id*, *namespace*, *request*, *resources=None*, *override_values=None*)
Bases: *onapsdk.msb.msb_service.MSB*

Instance class.

base_url = 'https://msb.api.simpledemo.onap.org:30283/api/multicloud-k8s/v1/v1/instanc

classmethod create(*cloud_region_id*, *profile_name*, *rb_name*, *rb_version*, *override_values=None*, *labels=None*)
Create Instance.

Parameters

- **cloud_region_id** (*str*) – Cloud region ID
- **profile_name** (*str*) – Name of profile to be instantiated
- **rb_name** (*str*) – (bytes): Definition name
- **rb_version** (*str*) – Definition version
- **override_values** (*dict*) – List of optional override values
- **labels** (*dict*) – List of optional labels

Returns Created object

Return type *Instance*

delete()
Delete Instance object.

Return type None

classmethod get_all()
Get all instantiated Kubernetes resources.

Yields *Instantiation* – Instantiation object

Return type Iterator[*Instance*]

classmethod get_by_id(*instance_id*)
Get Kubernetes resource by id.

Parameters **instance_id** (*str*) – instance ID

Returns Instantiation object

Return type *Instantiation*

property url
URL address.

Returns URL to Instance

Return type str

class onapsdk.msb.k8s.instance.**InstantiationParameter**(*name*, *value*)
Bases: object

Class to store instantiation parameters used to pass override_values and labels.

Contains two values: name of parameter and it's value

name: **str**

value: **str**

class onapsdk.msb.k8s.instance.**InstantiationRequest**(*request*)
Bases: object

Instantiation Request class.

Module contents

K8s package.

Submodules

onapsdk.msb.esr module

ESR module.

class onapsdk.msb.esr.**ESR**
Bases: *onapsdk.msb.msb_service.MSB*

External system EST module.

base_url = 'https://msb.api.simpledemo.onap.org:30283/api/aai-esr-server/v1/vims'

classmethod register_vim(*cloud_owner*, *cloud_region_id*, *cloud_type*, *cloud_region_version*, *auth_info_cloud_domain*, *auth_info_username*, *auth_info_password*, *auth_info_url*, *owner_defined_type=None*, *cloud_zone=None*, *physical_location_id=None*, *cloud_extra_info=None*, *auth_info_ssl_cacert=None*, *auth_info_ssl_insecure=None*)

Register VIM.

Parameters

- **cloud_owner** (*str*) – cloud owner name, can be customized, e.g. att-aic
- **cloud_region_id** (*str*) – cloud region info based on deployment, e.g. RegionOne
- **cloud_type** (*str*) – type of the cloud, decides which multicloud plugin to use, openstack or vio
- **cloud_region_version** (*str*) – cloud version, ocata, mitaka or other
- **auth_info_cloud_domain** (*str*) – domain info for keystone v3
- **auth_info_username** (*str*) – user name
- **auth_info_password** (*str*) – password
- **auth_info_url** (*str*) – authentication url of the cloud, e.g. keystone url
- **owner_defined_type** (*str, optional*) – cloud-owner defined type indicator (e.g., dcp, lcp). Defaults to None.
- **cloud_zone** (*str, optional*) – zone where the cloud is homed.. Defaults to None.
- **physical_location_id** (*str, optional*) – complex physical location id for cloud-region instance. Defaults to None.
- **cloud_extra_info** (*str, optional*) – extra info for Cloud. Defaults to None.
- **auth_info_ssl_cacert** (*str, optional*) – ca file content if enabled ssl on auth-url. Defaults to None.
- **auth_info_ssl_insecure** (*bool, optional*) – whether to verify VIM's certificate. Defaults to None.

Return type None

onapsdk.msb.msb_service module

Microsevice bus module.

class onapsdk.msb.msb_service.**MSB**

Bases: *onapsdk.onap_service.OnapService*

Microservice Bus base class.

base_url = 'https://msb.api.simpledemo.onap.org:30283'

headers: Dict[str, str] = {'Accept': 'application/json', 'Content-Type': 'applicati

onapsdk.msb.multicloud module

Multicloud module.

class onapsdk.msb.multicloud.**Multicloud**
Bases: *onapsdk.msb.msb_service.MSB*

MSB subclass to register/unregister instance to ONAP.

base_url = 'https://msb.api.simpledemo.onap.org:30283/api/multicloud/v1'

classmethod register_vim(*cloud_owner*, *cloud_region_id*, *default_tenant=None*)
Register a VIM instance to ONAP.

Parameters

- **cloud_owner** (*str*) – Cloud owner name
- **cloud_region_id** (*str*) – Cloud region ID
- **default_tenant** (*str, optional*) – Default tenant name. Defaults to None.

Return type None

classmethod unregister_vim(*cloud_owner*, *cloud_region_id*)
Unregister a VIM instance from ONAP.

Parameters

- **cloud_owner** (*str*) – Cloud owner name
- **cloud_region_id** (*str*) – Cloud region ID

Return type None

Module contents

Microsevice bus package.

onapsdk.nbi package

Submodules

onapsdk.nbi.nbi module

NBI module.

class onapsdk.nbi.nbi.**Nbi**
Bases: *onapsdk.onap_service.OnapService*, abc.ABC

NBI base class.

api_version = '/nbi/api/v4'

base_url = 'https://nbi.api.simpledemo.onap.org:30274'

classmethod is_status_ok()
Check NBI service status.

Returns True if NBI works fine, False otherwise

Return type bool

class onapsdk.nbi.nbi.**Service**(*name*, *service_id*, *service_specification_name*, *service_specification_id*, *customer_id*, *customer_role*, *href*)

Bases: *onapsdk.nbi.nbi.Nbi*

NBI service.

property customer

Service order Customer object.

Returns Customer object

Return type *Customer*

classmethod get_all(*customer_id='generic'*)

Get all services for selected customer.

Parameters **customer_id** (*str*) – Global customer ID

Yields *Service* – Service object

Return type Iterator[*Service*]

property service_specification

Service specification.

Returns Service specification object

Return type *ServiceSpecification*

class onapsdk.nbi.nbi.**ServiceOrder**(*unique_id*, *href*, *priority*, *description*, *category*, *external_id*, *service_instance_name*, *state=None*, *customer=None*, *customer_id=None*, *service_specification=None*, *service_specification_id=None*)

Bases: *onapsdk.nbi.nbi.Nbi*, *onapsdk.utils.mixins.WaitForFinishMixin*

Service order class.

class StatusEnum(*value*)

Bases: enum.Enum

Status enum.

Store possible statuses for service order:

- completed,
- failed,
- inProgress.

If instantiation has status which is not covered by these values *unknown* value is used.

ACKNOWLEDGED = 'acknowledged'

COMPLETED = 'completed'

FAILED = 'failed'

IN_PROGRESS = 'inProgress'

REJECTED = 'rejected'

UNKNOWN = 'unknown'

WAIT_FOR_SLEEP_TIME = 10

property completed
Store an information if service order is completed or not.

Service orded is completed if it's status is COMPLETED.

Returns True if service orded is completed, False otherwise.

Return type bool

classmethod create(*customer*, *service_specification*, *name=None*, *external_id=None*)
Create service order.

Returns ServiceOrder object

Return type *ServiceOrder*

property customer
Get customer object used in service order.

Returns Customer object

Return type *Customer*

property failed
Store an information if service order is failed or not.

Service orded is completed if it's status is FAILED.

Returns True if service orded is failed, False otherwise.

Return type bool

property finished
Store an information if service order is finished or not.

Service orded is finished if it's status is not ACKNOWLEDGED or IN_PROGRESS.

Returns True if service orded is finished, False otherwise.

Return type bool

classmethod get_all()
Get all service orders.

Returns ServiceOrder object

Return type Iterator[*ServiceOrder*]

property rejected
Store an information if service order is rejected or not.

Service orded is completed if it's status is REJECTED.

Returns True if service orded is rejected, False otherwise.

Return type bool

property service_specification
Service order service specification used in order item.

Returns Service specification

Return type *ServiceSpecification*

property status
Service order instantiation status.

It's populated by call Service order endpoint.

Returns Service order status.

Return type *StatusEnum*

class onapsdk.nbi.nbi.**ServiceSpecification**(*unique_id*, *name*, *invariant_uuid*, *category*, *distribution_status*, *version*, *lifecycle_status*)

Bases: *onapsdk.nbi.nbi.Nbi*

NBI service specification class.

classmethod get_all()

Get all service specifications.

Yields *ServiceSpecification* – Service specification object

Return type Iterator[*ServiceSpecification*]

classmethod get_by_id(*service_specification_id*)

Get service specification by ID.

Parameters **service_specification_id** (*str*) – Service specification ID

Returns Service specification object

Return type *ServiceSpecification*

Module contents

NBI package.

onapsdk.sdc package

Submodules

onapsdk.sdc.category_management module

SDC category management module.

class onapsdk.sdc.category_management.**BaseCategory**(*name*)

Bases: *onapsdk.sdc.SDC*, abc.ABC

Base SDC category class.

It's SDC admin resource, has no common properties with SDC resourcer or elements, so SDC class can't be it's base class.

SDC_ADMIN_USER = 'demo'

abstract classmethod category_name()

Class category name.

Used for logs.

Returns Category name

Return type str

classmethod create(*name*)

Create category instance.

Checks if category with given name exists and if it already exists just returns category with given name.

Returns Created category instance

Return type *BaseCategory*

classmethod get(*name*)
Get category with given name.

Raises ***ResourceNotFound*** – Category with given name does not exist

Returns BaseCategory instance

Return type *BaseCategory*

classmethod get_all(***kwargs*)
Get the categories list created in SDC.

Return type List[*SDC*]

Returns the list of the categories

classmethod headers()
Headers used for category management.

It uses SDC admin user.

Returns Headers

Return type Dict[str, str]

classmethod import_from_sdc(*values*)
Import category object from SDC.

Parameters **values** (*Dict[str, Any]*) – dict to parse returned from SDC.

Return type *BaseCategory*

class onapsdk.sdc.category_management.**ResourceCategory**(*name*)
Bases: *onapsdk.sdc.category_management.BaseCategory*

Resource category class.

classmethod category_name()
Resource category name.

Used for logging.

Returns Resource category name

Return type str

classmethod get(*name*, *subcategory=None*)
Get resource category with given name.

It returns resource category with all subcategories by default. You can get resource category with only one subcategory if you provide it's name as *subcategory* parameter.

Parameters

- **name** (*str*) – Resource category name.
- **subcategory** (*str, optional*) – Name of subcategory. Defaults to None.

Raises ***ResourceNotFound*** – Subcategory with given name does not exist

Returns BaseCategory instance

Return type *BaseCategory*

class onapsdk.sdc.category_management.**ServiceCategory**(*name*)
Bases: *onapsdk.sdc.category_management.BaseCategory*

Service category class.

classmethod category_name()
Service category name.

Used for logging.

Returns Service category name

Return type str

onapsdk.sdc.component module

SDC Component module.

class onapsdk.sdc.component.**Component**(*created_from_csar*, *actual_component_uid*, *unique_id*, *normalized_name*, *name*, *origin_type*, *customization_uuid*, *component_uid*, *component_version*, *tosca_component_name*, *component_name*, *group_instances*, *sdc_resource*, *parent_sdc_resource*)
Bases: object

Component dataclass.

actual_component_uid: **str**

component_name: **str**

component_uid: **str**

component_version: **str**

classmethod create_from_api_response(*api_response*, *sdc_resource*, *parent_sdc_resource*)
Create component from api response.

Parameters

- **api_response** (*Dict[str, Any]*) – component API response
- **sdc_resource** (SdcResource) – component's SDC resource
- **parent_sdc_resource** (SdcResource) – component's parent SDC resource

Returns Component created using api_response and SDC resource

Return type *Component*

created_from_csar: **bool**

customization_uuid: **str**

delete()
Delete component.

Return type None

get_property(*property_name*)
Get component property by it's name.

Parameters **property_name** (*str*) – property name

Raises ***ParameterError*** – Component has no property with given name

Returns Component's property object

Return type *ComponentProperty*

group_instances: Optional[List[Dict[str, Any]]]

name: str

normalized_name: str

origin_type: str

parent_sdc_resource: SdcResource

property properties

Component properties.

In SDC it's named as properties, but we uses "inputs" endpoint to fetch them. Structure is also input's like, but it's a property.

Yields *ComponentProperty* – Component property object

Return type Iterator[*ComponentProperty*]

property properties_url

Url to get component's properties.

Returns Compoent's properties url

Return type str

property properties_value_url

Url to set component property value.

Returns Url to set component property value

Return type str

sdc_resource: SdcResource

set_property_value(*property_obj*, *value*)

Set property value.

Set given value to component property

Parameters

- **property_obj** (ComponentProperty) – Component property object
- **value** (*Any*) – Property value to set

Return type None

tosca_component_name: str

unique_id: str

onapsdk.sdc.pnf module

Pnf module.

class onapsdk.sdc.pnf.**Pnf**(*name=None*, *version=None*, *vendor=None*, *sdc_values=None*, *vsp=None*, *properties=None*, *inputs=None*, *category=None*, *subcategory=None*)

Bases: *onapsdk.sdc.sdc_resource.SdcResource*

ONAP PNF Object used for SDC operations.

name
the name of the pnf. Defaults to "ONAP-test-PNF".

Type str

identifier
the unique ID of the pnf from SDC.

Type str

status
the status of the pnf from SDC.

Type str

version
the version ID of the vendor from SDC.

Type str

uuid
the UUID of the PNF (which is different from identifier, don't ask why...)

Type str

unique_identifier
Yet Another ID, just to puzzle us...

Type str

vendor
the vendor of the PNF

Type optional

vsp
the vsp related to the PNF

Type optional

create()
Create the PNF in SDC if not already existing.

Return type None

onapsdk.sdc.properties module

Service properties module.

class onapsdk.sdc.properties.**ComponentProperty**(*unique_id*, *property_type*, *name*, *component*, *_value=None*)

Bases: object

Component property dataclass.

Component properties are inputs objects in SDC, but in logic it's a property.

component: **Component**

name: **str**

property_type: **str**

unique_id: **str**

property value
Property value getter.

Returns Property value

Return type Any

class onapsdk.sdc.properties.**Input**(*unique_id*, *input_type*, *name*, *sdc_resource*, *_default_value=None*)

Bases: object

Property input dataclass.

property default_value
Input default value.

Returns Input default value

Return type Any

input_type: **str**

name: **str**

sdc_resource: **SdcResource**

unique_id: **str**

class onapsdk.sdc.properties.**NestedInput**(*sdc_resource*, *input_obj*)

Bases: object

Dataclass used for nested input declaration.

input_obj: ***onapsdk.sdc.properties.Input***

sdc_resource: **SdcResource**

class onapsdk.sdc.properties.**Property**(*name*, *property_type*, *description=None*, *unique_id=None*, *parent_unique_id=None*, *sdc_resource=None*, *value=None*, *get_input_values=None*)

Bases: object

Service property class.

property input
Property input.

Returns property Input object. Returns None if property has no associated input.

Raises

- ***ParameterError*** – Input has no associated SdcResource
- ***ParameterError*** – Input for given property does not exits. It shouldn't ever happen, but it's possible if after you get property object someone delete input.

Returns Property input object.

Return type *Input*

property value
Value property.

Get property value.

Returns Property value

Return type Any

onapsdk.sdc.sdc_element module

SDC Element module.

class onapsdk.sdc.sdc_element.**SdcElement**(*name=None*)
Bases: *onapsdk.sdc.SdcOnboardable*, abc.ABC

Mother Class of all SDC elements.

ACTION_METHOD: str = 'PUT'

ACTION_TEMPLATE: str = 'sdc_element_action.json.j2'

abstract classmethod import_from_sdc(*values*)
Import SdcElement from SDC.

Parameters **values** (*Dict[str, Any]*) – dict to parse returned from SDC.

Raises **NotImplementedError** – this is an abstract method.

Return type *SdcElement*

load()
Load Object information from SDC.

Return type None

update_informations_from_sdc(*details*)
Update instance with details from SDC.

Parameters **details** (*[type]*) – [description]

Return type None

update_informations_from_sdc_creation(*details*)
Update instance with details from SDC after creation.

Parameters **details** (*[type]*) – the details from SDC

Return type None

onapsdk.sdc.sdc_resource module

SDC Element module.

class onapsdk.sdc.sdc_resource.**SdcResource**(*name=None*, *version=None*, *sdc_values=None*, *properties=None*, *inputs=None*, *category=None*, *subcategory=None*)

Bases: *onapsdk.sdc.SdcOnboardable*, abc.ABC

Mother Class of all SDC resources.

ACTION_METHOD: str = 'POST'

ACTION_TEMPLATE: str = 'sdc_resource_action.json.j2'

RESOURCE_PATH = 'resources'

add_deployment_artifact(*artifact_type*, *artifact_label*, *artifact_name*, *artifact*)

Add deployment artifact to resource.

Add deployment artifact to resource using payload data.

Parameters

- **artifact_type** (*str*) – all SDC artifact types are supported (DCAE_*, HEAT_*, . . .)
- **artifact_name** (*str*) – the artifact file name including its extension
- **artifact** (*str*) – artifact file to upload
- **artifact_label** (*str*) – Unique Identifier of the artifact within the VF / Service.

Raises ***StatusError*** – Resource has not DRAFT status

property add_deployment_artifacts_url

Add deployment artifacts url.

Returns Url used to add deployment artifacts

Return type str

add_property(*property_to_add*)

Add property to resource.

Call SDC FE API to add property to resource.

Parameters **property_to_add** (Property) – Property object to add to resource.

Raises ***StatusError*** – Resource has not DRAFT status

Return type None

property add_property_url

Add property url.

Returns Url used to add property

Return type str

add_resource(*resource*)

Add a Resource.

Parameters **resource** (SdcResource) – the resource to add

Return type None

property category
Sdc resource category.

Depends on the resource type returns ResourceCategory or ServiceCategory.

Returns resource category

Return type Uniton[*ResourceCategory*, *ServiceCategory*]

certify()
Certify SDC resource.

Return type None

checkout()
Checkout SDC resource.

Return type None

property components
Resource components.

Iterate resource components.

Yields *Component* – Resource component object

Return type Iterator[*Component*]

create()
Create resource.

Abstract method which should be implemented by subclasses and creates resource in SDC.

Raises **NotImplementedError** – Method not implemented by subclasses.

Return type None

declare_input(*input_to_declare*)
Declare input for given property or nested input object.

Call SDC FE API to declare input for given property.

Parameters **input_declaration** (*Union[Property, NestedInput]*) – Property to declare input or NestedInput object

Raises ***ParameterError*** – if the given property is not SDC resource property

Return type None

declare_input_for_own_property(*property_obj*)
Declare input for resource's property.

For each property input can be declared.

Parameters **property_obj** (Property) – Property to declare input

Return type None

declare_nested_input(*nested_input*)
Declare nested input for SDC resource.

Nested input is an input of one of the components.

Parameters **nested_input** (NestedInput) – Nested input object

Return type None

deep_load()
Deep load Object informations from SDC.

Return type None

property deployment_artifacts_url
Deployment artifacts url.

Returns SdcResource Deployment artifacts url

Return type str

get_category_for_new_resource()
Get category for resource not created in SDC yet.

If no category values are provided default category is going to be used.

Returns Category of the new resource

Return type *ResourceCategory*

get_component(*sdc_resource*)
Get resource's component.

Get component by SdcResource object.

Parameters **sdc_resource** (SdcResource) – Component's SdcResource

Raises ***ResourceNotFound*** – Component with given SdcResource does not exist

Returns Component object

Return type *Component*

get_component_by_name(*component_name*)
Get resource's component by it's name.

Get component by name.

Parameters **component_name** (*str*) – Component's name

Raises ***ResourceNotFound*** – Component with given name does not exist

Returns Component object

Return type *Component*

get_component_properties_url(*component*)
Url to get component's properties.

This method is here because component can have different url when it's a component of another SDC resource type, eg. for service and for VF components have different urls.

Parameters **component** (Component) – Component object to prepare url for

Returns Component's properties url

Return type str

get_component_properties_value_set_url(*component*)
Url to set component property value.

This method is here because component can have different url when it's a component of another SDC resource type, eg. for service and for VF components have different urls.

Parameters **component** (Component) – Component object to prepare url for

Returns Component's properties url

Return type str

get_input(*input_name*)
Get input by it's name.

Parameters **input_name** (*str*) – Input name

Raises ***ResourceNotFound*** – Resource doesn't have input with given name

Returns Found input object

Return type *Input*

get_property(*property_name*)
Get resource property by it's name.

Parameters **property_name** (*str*) – property name

Raises ***ResourceNotFound*** – Resource has no property with given name

Returns Resource's property object

Return type *Property*

headers: Dict[str, str] = {'Accept': 'application/json', 'Authorization': 'Basic YW

classmethod import_from_sdc(*values*)
Import SdcResource from SDC.

Parameters **values** (*Dict[str, Any]*) – dict to parse returned from SDC.

Returns the created resource

Return type *SdcResource*

property inputs
SDC resource inputs.

Iterate resource inputs.

Yields *Iterator[Input]* – Resource input

Return type Iterator[*Input*]

is_own_property(*property_to_check*)
Check if given property is one of the resource's properties.

Parameters **property_to_check** (Property) – Property to check

Returns True if resource has given property, False otherwise

Return type bool

load()
Load Object information from SDC.

Return type None

onboard()
Onboard resource in SDC.

Return type None

property origin_type
Resource origin type.

Value needed for composition. It's used for adding SDC resource as an another SDC resource component.

Returns SDC resource origin type

Return type str

property properties
SDC resource properties.

Iterate resource properties.

Yields *Property* – Resource property

Return type Iterator[*Property*]

property properties_url
Properties url.

Returns SdcResource properties url

Return type str

property resource_inputs_url
Resource inputs url.

Method which returns url which point to resource inputs.

Returns Resource inputs url

Return type str

set_input_default_value(*input_obj*, *default_value*)
Set input default value.

Set given value as input default value

Parameters

- **input_obj** (Input) – Input object
- **value** (*Any*) – Default value to set

Return type None

property set_input_default_value_url
Url to set input default value.

Returns SDC API url used to set input default value

Return type str

set_property_value(*property_obj*, *value*)
Set property value.

Set given value to resource property

Parameters

- **property_obj** (Property) – Property object
- **value** (*Any*) – Property value to set

Raises ***ParameterError*** – if the given property is not the resource's property

Return type None

undo_checkout()
Undo Checkout SDC resource.

Return type None

property unique_identifier
Return and lazy load the unique_identifier.

Return type str

property unique_uuid
Return and lazy load the unique_uuid.

Return type str

update_informations_from_sdc(*details*)
Update instance with details from SDC.

Parameters **details** (*[type]*) – [description]

Return type None

update_informations_from_sdc_creation(*details*)
Update instance with details from SDC after creation.

Parameters **details** (*[type]*) – the details from SDC

Return type None

onapsdk.sdc.service module

Service module.

class onapsdk.sdc.service.**Network**(*name*, *node_template_type*, *model_name*, *model_version_id*, *model_invariant_id*, *model_version*, *model_customization_id*, *model_instance_name*, *component*)
Bases: *onapsdk.sdc.service.NodeTemplate*

Network dataclass.

component: Component

model_customization_id: str

model_instance_name: str

model_invariant_id: str

model_name: str

model_version: str

model_version_id: str

name: str

node_template_type: str

class onapsdk.sdc.service.**NodeTemplate**(*name*, *node_template_type*, *model_name*, *model_version_id*, *model_invariant_id*, *model_version*, *model_customization_id*, *model_instance_name*, *component*)
Bases: object

Node template dataclass.

Base class for Vnf, Pnf and Network classes.

component: **Component**

model_customization_id: **str**

model_instance_name: **str**

model_invariant_id: **str**

model_name: **str**

model_version: **str**

model_version_id: **str**

name: **str**

node_template_type: **str**

property properties
Node template properties.

Returns Node template properties iterator

Return type Iterator[*Property*]

class onapsdk.sdc.service.**Pnf**(*name*, *node_template_type*, *model_name*, *model_version_id*, *model_invariant_id*, *model_version*, *model_customization_id*, *model_instance_name*, *component*)

Bases: *onapsdk.sdc.service.NodeTemplate*

Pnf dataclass.

component: **Component**

model_customization_id: **str**

model_instance_name: **str**

model_invariant_id: **str**

model_name: **str**

model_version: **str**

model_version_id: **str**

name: **str**

node_template_type: **str**

class onapsdk.sdc.service.**Service**(*name=None*, *version=None*, *sdc_values=None*, *resources=None*, *properties=None*, *inputs=None*, *instantiation_type=None*, *category=None*, *role=''*, *function=''*, *service_type=''*)

Bases: *onapsdk.sdc.sdc_resource.SdcResource*

ONAP Service Object used for SDC operations.

name
the name of the service. Defaults to “ONAP-test-Service”.

Type str

identifier
the unique ID of the service from SDC.

Type str

status
the status of the service from SDC.

Type str

version
the version ID of the service from SDC.

Type str

uuid
the UUID of the Service (which is different from identifier, don't ask why...)

Type str

distribution_status
the status of distribution in the different ONAP parts.

Type str

distribution_id
the ID of the distribution when service is distributed.

Type str

distributed
True if the service is distributed

Type bool

unique_identifier
Yet Another ID, just to puzzle us...

Type str

SERVICE_PATH = 'services'

add_artifact_to_vf(*vnf_name*, *artifact_type*, *artifact_name*, *artifact=None*)
Add artifact to vf.

Add artifact to vf using payload data.

Raises ***RequestError*** – file upload (POST request) for an artifact fails.

Parameters

- **vnf_name** (*str*) – the vnf which we want to add the artifact
- **artifact_type** (*str*) – all SDC artifact types are supported (DCAE_*, HEAT_*, ...)
- **artifact_name** (*str*) – the artifact file name including its extension
- **artifact** (*str*) – binary data to upload

property add_deployment_artifacts_url
Add deployment artifacts url.

Returns Url used to add deployment artifacts

Return type str

approve()
Approve Service in SDC.

Return type None

certify()
Certify Service in SDC.

Return type None

checkin()
Checkin Service.

Return type None

create()
Create the Service in SDC if not already existing.

Return type None

create_node_template(*node_template_type*, *component*)
Create a node template type object.

The base of the all node template types objects (Vnf, Pnf, Network) is the same. The difference is only for the Vnf which can have vf modules associated with. Vf modules could have "vf_module_label" property with"base_template_dummy_ignore" value. These vf modules should be ignored/

Parameters

- **node_template_type** (*Type[NodeTemplate]*) – Node template class type
- **component** (Component) – Component on which base node template object should be created

Returns Node template object created from component

Return type *NodeTemplate*

declare_resources_and_properties()
Delcare resources and properties.

It declares also inputs.

Return type None

property deployment_artifacts_url
Deployment artifacts url.

Returns SdcResource Deployment artifacts url

Return type str

distribute()
Apptove Service in SDC.

Return type None

property distributed
Return and lazy load the distributed state.

Return type bool

property distribution_id
Return and lazy load the distribution_id.

Return type str

classmethod get_by_unique_uuid(*unique_uuid*)
Get the service model using unique uuid.

Returns object with provided unique_uuid

Return type *Service*

Raises ***ResourceNotFound*** – No service with given unique_uuid exists

get_category_for_new_resource()
Get category for service not created in SDC yet.

If no category values are provided default category is going to be used.

Returns Category of the new service

Return type *ServiceCategory*

get_component_properties_url(*component*)
Url to get component's properties.

This method is here because component can have different url when it's a component of another SDC resource type, eg. for service and for VF components have different urls. Also for VL origin type components properties url is different than for the other types.

Parameters **component** (Component) – Component object to prepare url for

Returns Component's properties url

Return type str

get_component_properties_value_set_url(*component*)
Url to set component property value.

This method is here because component can have different url when it's a component of another SDC resource type, eg. for service and for VF components have different urls. Also for VL origin type components properties url is different than for the other types.

Parameters **component** (Component) – Component object to prepare url for

Returns Component's properties url

Return type str

get_nf_unique_id(*nf_name*)
Get nf (network function) uniqueID.

Get nf uniqueID from service nf in sdc.

Parameters **nf_name** (*str*) – the nf from which we extract the unique ID

Return type str

Returns the nf unique ID

Raises ***ResourceNotFound*** – Couldn't find NF by name.

get_tosca()
Get Service tosca files and save it.

Return type None

property has_pnfs
Check if service has at least one PNF component.

Return type bool

property has_vls
Check if service has at least one VL component.

Return type bool

property has_vnfs
Check if service has at least one VF component.

Return type bool

property instantiation_type
Service instantiation type.

One of *ServiceInstantiationType* enum value.

Returns Service instantiation type

Return type *ServiceInstantiationType*

load_metadata()
Load Metada of Service and retrieve informations.

Return type None

property metadata_url
Metadata url.

Returns Service metadata url

Return type str

property networks
Service networks.

Load networks from service's components generator.

Returns Network objects generator

Return type Iterator[*Network*]

onboard()
Onboard the Service in SDC.

Raises

- ***StatusError*** – service has an invalid status
- ***ParameterError*** – no resources, no properties for service in DRAFT status

Return type None

property origin_type
Service origin type.

Value needed for composition. It's used for adding SDC resource as an another SDC resource component. For Service that value has to be set to "ServiceProxy".

Returns Service resource origin type

Return type str

property pnfs
Service Pnfs.

Load PNFS from components generator.

Returns Pnf objects generator

Return type Iterator[*Pnf*]

property properties_url
Properties url.

Returns SdcResource properties url

Return type str

redistribute()
Apptove Service in SDC.

Return type None

property resource_inputs_url
Service inputs url.

Returns Service inputs url

Return type str

property set_input_default_value_url
Url to set input default value.

Returns SDC API url used to set input default value

Return type str

start_certification()
Start Certification on Service.

Return type None

submit()
Really submit the SDC Service.

Return type None

property tosca_model
Service's tosca model file.

Send request to get service TOSCA model,

Returns TOSCA model file bytes

Return type bytes

property tosca_template
Service tosca template file.

Get tosca template from service tosca model bytes.

Returns Tosca template file

Return type str

property vnfs
Service Vnfs.

Load VNFs from components generator. It creates a generator of the vf modules as well, but without

vf modules which has "vf_module_label" property value equal to "base_template_dummy_ignore".

Returns Vnf objects iterator

Return type Iterator[*Vnf*]

class onapsdk.sdc.service.**ServiceInstantiationType**(*value*)
Bases: enum.Enum

Service instantiation type enum class.

Service can be instantiated using *A-la-carte* or *Macro* flow. It has to be determined during design time. That class stores these two values to set during initialization.

A_LA_CARTE = 'A-la-carte'

MACRO = 'Macro'

class onapsdk.sdc.service.**VfModule**(*name*, *group_type*, *model_name*, *model_version_id*, *model_invariant_uuid*, *model_version*, *model_customization_id*, *properties*)
Bases: object

VfModule dataclass.

group_type: str

model_customization_id: str

model_invariant_uuid: str

model_name: str

model_version: str

model_version_id: str

name: str

properties: Iterator[*onapsdk.sdc.properties.Property*]

class onapsdk.sdc.service.**Vnf**(*name*, *node_template_type*, *model_name*, *model_version_id*, *model_invariant_id*, *model_version*, *model_customization_id*, *model_instance_name*, *component*, *vf_modules=<factory>*)
Bases: *onapsdk.sdc.service.NodeTemplate*

Vnf dataclass.

vf_modules: List[*onapsdk.sdc.service.VfModule*]

onapsdk.sdc.vendor module

Vendor module.

class onapsdk.sdc.vendor.**Vendor**(*name=None*)
Bases: *onapsdk.sdc.sdc_element.SdcElement*

ONAP Vendor Object used for SDC operations.

name
the name of the vendor. Defaults to "Generic-Vendor".

Type str

identifier
the unique ID of the vendor from SDC.

Type str

status
the status of the vendor from SDC.

Type str

version
the version ID of the vendor from SDC.

Type str

VENDOR_PATH = 'vendor-license-models'

create()
Create the vendor in SDC if not already existing.

Return type None

headers: Dict[str, str] = {'Accept': 'application/json', 'Authorization': 'Basic YW

classmethod import_from_sdc(*values*)
Import Vendor from SDC.

Parameters **values** (*Dict[str, Any]*) – dict to parse returned from SDC.

Return type *Vendor*

Returns a Vsp instance with right values

onboard()
Onboard the vendor in SDC.

Return type None

submit()
Submit the SDC vendor in order to enable it.

Return type None

update_informations_from_sdc(*details*)
Update instance with details from SDC.

Parameters **details** (*Dict[str, Any]*) – dict from SDC

Return type None

onapsdk.sdc.vf module

Vf module.

class onapsdk.sdc.vf.**Vf**(*name=None*, *version=None*, *sdc_values=None*, *vsp=None*, *properties=None*, *inputs=None*, *category=None*, *subcategory=None*, *vendor=None*)
Bases: *onapsdk.sdc.sdc_resource.SdcResource*

ONAP Vf Object used for SDC operations.

name
the name of the vendor. Defaults to “Generic-Vendor”.

Type str

identifier
the unique ID of the vendor from SDC.

Type str

status
the status of the vendor from SDC.

Type str

version
the version ID of the vendor from SDC.

Type str

vsp
the Vsp used for VF creation

Type *Vsp*

uuid
the UUID of the VF (which is different from identifier, don't ask why...)

Type str

unique_identifier
Yet Another ID, just to puzzle us...

Type str

create()
Create the Vf in SDC if not already existing.

Raises ***ParameterError*** – VSP is not provided during VF object initalization

Return type None

declare_input(*input_to_declare*)
Declare input for given property, nested input or component property object.

Call SDC FE API to declare input for given property.

Parameters **input_declaration** (*Union[Property, NestedInput]*) – Property or ComponentProperty to declare input or NestedInput object

Raises ***ParameterError*** – if the given property is not SDC resource property

Return type None

update_vsp(*vsp*)
Update Vsp.

Update VSP UUID and version for Vf object.

Parameters **vsp** (Vsp) – Object to be used in Vf

Raises ***ValidationError*** – Vf object request has invalid structure.

Return type None

property vendor
Vendor related wth Vf.

If Vf is created vendor is get from it's resource metadata. Otherwise it's vendor provided by the user or the vendor from vsp. It's possible that method returns None, but it won't be possible then

to create that Vf resource.

Returns Vendor object related with Vf

Return type Optional[*Vendor*]

onapsdk.sdc.vl module

Vl module.

class onapsdk.sdc.vl.**Vl**(*name*, *version=None*, *sdc_values=None*)
Bases: *onapsdk.sdc.sdc_resource.SdcResource*

ONAP Vl Object used for SDC operations.

onapsdk.sdc.vsp module

VSP module.

class onapsdk.sdc.vsp.**Vsp**(*name=None*, *package=None*, *vendor=None*)
Bases: *onapsdk.sdc.sdc_element.SdcElement*

ONAP VSP Object used for SDC operations.

name
the name of the vsp. Defaults to "ONAP-test-VSP".

Type str

identifier
the unique ID of the VSP from SDC.

Type str

status
the status of the VSP from SDC.

Type str

version
the version ID of the VSP from SDC.

Type str

csar_uuid
the CSAR ID of the VSP from SDC.

Type str

vendor
The VSP Vendor

Type *Vendor*

VSP_PATH = 'vendor-software-products'

commit()
Commit the SDC Vsp.

Return type None

create()
Create the Vsp in SDC if not already existing.

Return type None

create_csar()
Create the CSAR package in the SDC Vsp.

Return type None

create_new_version()
Create new version of VSP.

Create a new major version of VSP so it would be possible to update a package or do some changes in VSP.

Return type None

property csar_uuid
Return and lazy load the CSAR UUID.

Return type str

headers: Dict[str, str] = {'Accept': 'application/json', 'Authorization': 'Basic YW

classmethod import_from_sdc(*values*)
Import Vsp from SDC.

Parameters **values** (*Dict[str, Any]*) – dict to parse returned from SDC.

Return type *Vsp*

Returns a Vsp instance with right values

load_status()
Load Vsp status from SDC.

rules are following:

- DRAFT: status == DRAFT and networkPackageName not present
- UPLOADED: status == DRAFT and networkPackageName present and validationData not present
- VALIDATED: status == DRAFT and networkPackageName present and validationData present and state.dirty = true
- COMMITED: status == DRAFT and networkPackageName present and validationData present and state.dirty = false
- CERTIFIED: status == CERTIFIED

status is found in sdc items state is found in sdc version from items networkPackageName and validationData is found in SDC vsp show

Return type None

onboard()
Onboard the VSP in SDC.

Return type None

property status
Return and load the status.

submit()
Submit the SDC Vsp in order to enable it.

Return type None

update_package(*package_to_upload*)
Upload given zip file into SDC as artifacts for this Vsp.

Parameters **package_to_upload** (*file*) – the zip file to upload

Return type None

upload_package(*package_to_upload*)
Upload given zip file into SDC as artifacts for this Vsp.

Parameters **package_to_upload** (*file*) – the zip file to upload

Return type None

validate()
Validate the artifacts uploaded.

Return type None

property vendor
Return and lazy load the vendor.

Return type *Vendor*

Module contents

SDC Element module.

class onapsdk.sdc.**SDC**(*name=None*)
Bases: *onapsdk.onap_service.OnapService*, abc.ABC

Mother Class of all SDC elements.

base_back_url = 'https://sdc.api.be.simpledemo.onap.org:30204'

base_front_url = 'https://sdc.api.fe.simpledemo.onap.org:30207'

exists()
Check if object already exists in SDC and update infos.

Return type bool

Returns True if exists, False either

classmethod get_all(***kwargs*)
Get the objects list created in SDC.

Return type List[*SDC*]

Returns the list of the objects

classmethod get_guis()
Retrieve the status of the SDC GUIs.

Only one GUI is referenced for SDC the SDC Front End

Return the list of GUIs

Return type GuiItem

abstract classmethod import_from_sdc(*values*)
Import Sdc object from SDC.

Parameters **values** (*Dict[str, Any]*) – dict to parse returned from SDC.

Raises **NotImplementedError** – this is an abstract method.

Return type *SDC*

server: str = 'SDC'

class onapsdk.sdc.**SdcOnboardable**(*name=None*)

Bases: *onapsdk.sdc.SDC*, abc.ABC

Base class for onboardable SDC resources (Vendors, Services, . . .).

ACTION_METHOD: str

ACTION_TEMPLATE: str

created()

Determine if SDC is created.

Return type bool

property identifier

Return and lazy load the identifier.

Return type str

abstract load()

Load Object information from SDC.

Raises **NotImplementedError** – this is an abstract method.

Return type None

abstract onboard()

Onboard resource.

Onboarding is a full stack of actions which needs to be done to make SDC resource ready to use. It depends on the type of object but most of them needs to be created and submitted.

Return type None

property status

Return and lazy load the status.

Return type str

submit()

Submit the SDC object in order to enable it.

Return type None

abstract update_informations_from_sdc(*details*)

Update instance with details from SDC.

Parameters **details** (*[type]*) – [description]

Return type None

abstract update_informations_from_sdc_creation(*details*)

Update instance with details from SDC after creation.

Parameters **details** (*[type]*) – the details from SDC

Return type None

property version

Return and lazy load the version.

Return type str

onapsdk.sdnc package

Submodules

onapsdk.sdnc.preload module

SDNC preload module.

class onapsdk.sdnc.preload.**NetworkPreload**
Bases: *onapsdk.sdnc.preload.Preload*

Class to upload network module preload.

classmethod upload_network_preload(*network*, *network_instance_name*, *subnets=None*)
Upload network preload.

Parameters

- **network** (Network) – Network object
- **network_instance_name** (*str*) – network instance name
- **subnets** (*Iterable[Subnet], optional*) – Iterable object of Subnet. Defaults to None.

Return type None

class onapsdk.sdnc.preload.**Preload**
Bases: *onapsdk.sdnc.sdnc_element.SdncElement*

Preload base class.

headers: Dict[str, str] = {'Accept': 'application/json', 'Content-Type': 'applicati

class onapsdk.sdnc.preload.**PreloadInformation**(*preload_id*, *preload_type*, *preload_data*)
Bases: *onapsdk.sdnc.preload.Preload*

Preload information.

classmethod get_all()
Get all preload informations.

Get all uploaded preloads.

Yields *PreloadInformation* – Preload information object

Return type Iterable[*PreloadInformation*]

class onapsdk.sdnc.preload.**VfModulePreload**
Bases: *onapsdk.sdnc.preload.Preload*

Class to upload vf module preload.

classmethod upload_vf_module_preload(*vnf_instance*, *vf_module_instance_name*, *vf_module*, *vnf_parameters=None*)
Upload vf module preload.

Parameters

- **vnf_instance** (VnfInstance) – VnfInstance object
- **vf_module_instance_name** (*str*) – VF module instance name
- **vf_module** (VfModule) – VF module

- **vnf_parameters** (*Iterable[InstantiationParameter], optional*) – Iterable object of InstantiationParameter. Defaults to None.

Return type None

onapsdk.sdnc.sdnc_element module

SDNC base module.

class onapsdk.sdnc.sdnc_element.**SdncElement**
Bases: *onapsdk.onap_service.OnapService*

SDNC base class.

base_url = 'https://sdnc.api.simpledemo.onap.org:30267'

classmethod get_guis()
Retrieve the status of the SDNC GUIs.

There are 2 GUIS - SDNC DG Builder - SDNC ODL

Return the list of GUIs

Return type GuiItem

Module contents

SDNC package.

onapsdk.so package

Submodules

onapsdk.so.deletion module

Deletion module.

class onapsdk.so.deletion.**DeletionRequest**(*request_id*)
Bases: *onapsdk.so.so_element.OrchestrationRequest*, abc.ABC

Deletion request base class.

classmethod send_request(*instance*, *a_la_carte=True*)
Abstract method to send instance deletion request.

Raises **NotImplementedError** – Needs to be implemented in inheriting classes

Return type Deletion

class onapsdk.so.deletion.**NetworkDeletionRequest**(*request_id*)
Bases: *onapsdk.so.deletion.DeletionRequest*

Network deletion request class.

classmethod send_request(*instance*, *a_la_carte=True*)
Send request to SO to delete Network instance.

Parameters

- **instance** (NetworkInstance) – Network instance to delete

- **a_la_carte** (*boolean*) – deletion mode

Returns Deletion request object

Return type *NetworkDeletionRequest*

class onapsdk.so.deletion.**ServiceDeletionRequest**(*request_id*)
Bases: *onapsdk.so.deletion.DeletionRequest*

Service deletion request class.

classmethod send_request(*instance*, *a_la_carte=True*)
Send request to SO to delete service instance.

Parameters

- **instance** (ServiceInstance) – service instance to delete
- **a_la_carte** (*boolean*) – deletion mode

Returns Deletion request object

Return type *ServiceDeletionRequest*

class onapsdk.so.deletion.**VfModuleDeletionRequest**(*request_id*)
Bases: *onapsdk.so.deletion.DeletionRequest*

VF module deletion class.

classmethod send_request(*instance*, *a_la_carte=True*)
Send request to SO to delete VNF instance.

Parameters

- **instance** (VfModuleInstance) – Vf Module instance to delete
- **a_la_carte** (*boolean*) – deletion mode

Returns Deletion request object

Return type *VnfDeletionRequest*

class onapsdk.so.deletion.**VnfDeletionRequest**(*request_id*)
Bases: *onapsdk.so.deletion.DeletionRequest*

VNF deletion class.

classmethod send_request(*instance*, *a_la_carte=True*)
Send request to SO to delete VNF instance.

Parameters

- **instance** (VnfInstance) – VNF instance to delete
- **a_la_carte** (*boolean*) – deletion mode

Returns Deletion request object

Return type *VnfDeletionRequest*

onapsdk.so.instantiation module

Instantion module.

class onapsdk.so.instantiation.**Instantiation**(*name*, *request_id*, *instance_id*)
Bases: *onapsdk.so.so_element.OrchestrationRequest*, abc.ABC

Abstract class used for instantiation.

class onapsdk.so.instantiation.**InstantiationParameter**(*name*, *value*)
Bases: object

Class to store instantiation parameters used for preload or macro instantiation.

Contains two values: name of parameter and it's value

name: **str**

value: **str**

class onapsdk.so.instantiation.**NetworkInstantiation**(*name*, *request_id*, *instance_id*, *line_of_business*, *platform*, *network*)
Bases: *onapsdk.so.instantiation.NodeTemplateInstantiation*

Network instantiation class.

classmethod instantiate_ala_carte(*aai_service_instance*, *network_object*, *line_of_business*, *platform*, *cloud_region*, *tenant*, *network_instance_name=None*, *subnets=None*)
Instantiate Network using a'la carte method.

Parameters

- **network_object** (Network) – Network to instantiate
- **line_of_business** (*str*) – LineOfBusiness name to use in instantiation request
- **platform** (*str*) – Platform name to use in instantiation request
- **cloud_region** (CloudRegion) – Cloud region to use in instantiation request.
- **tenant** (Tenant) – Tenant to use in instnatiation request.
- **network_instance_name** (*str, optional*) – Network instance name. Defaults to None.

Returns NetworkInstantiation object

Return type *NetworkInstantiation*

class onapsdk.so.instantiation.**NodeTemplateInstantiation**(*name*, *request_id*, *instance_id*, *line_of_business*, *platform*)
Bases: *onapsdk.so.instantiation.Instantiation*, abc.ABC

Base class for service's node_template object instantiation.

class onapsdk.so.instantiation.**Operation**(*request_method*, *request_suffix*)
Bases: object

Operation class with data about method and suffix for VnfOperation.

request_method: **str**

request_suffix: **str**

class onapsdk.so.instantiation.**ServiceInstantiation**(*name*, *request_id*, *instance_id*, *sdc_service*, *cloud_region*, *tenant*, *customer*, *owning_entity*, *project*)

Bases: *onapsdk.so.instantiation.Instantiation*

Service instantiation class.

property aai_service_instance

Service instance associated with service instantiation request.

Raises

- ***StatusError*** – if a service is not instantiated - not in COMPLETE status.
- ***APIError*** – A&AI resource is not created

Returns ServiceInstance

Return type *ServiceInstance*

classmethod instantiate_ala_carte(*sdc_service*, *cloud_region*, *tenant*, *customer*, *owning_entity*, *project*, *service_subscription*, *service_instance_name=None*, *enable_multicloud=False*)

Instantiate service using SO a'la carte request.

Parameters

- **sdc_service** (*SdcService*) – Service to instantiate
- **cloud_region** (CloudRegion) – Cloud region to use in instantiation request
- **tenant** (Tenant) – Tenant to use in instantiation request
- **customer** (Customer) – Customer to use in instantiation request
- **owning_entity** (OwningEntity) – Owning entity to use in instantiation request
- **project** (*str*) – Project name to use in instantiation request
- **service_subscription** (ServiceSubscription) – Customer's service subsription.
- **service_instance_name** (*str, optional*) – Service instance name. Defaults to None.
- **enable_multicloud** (*bool, optional*) – Determines if Multicloud should be enabled for instantiation request. Defaults to False.

Raises ***StatusError*** – if a service is not distributed.

Returns instantiation request object

Return type *ServiceInstantiation*

classmethod instantiate_macro(*sdc_service*, *customer*, *owning_entity*, *project*, *line_of_business*, *platform*, *aai_service=None*, *cloud_region=None*, *tenant=None*, *service_instance_name=None*, *vnf_parameters=None*, *enable_multicloud=False*, *so_service=None*, *service_subscription=None*)

Instantiate service using SO macro request.

Parameters

- **sdc_service** (*SdcService*) – Service to instantiate
- **customer** (Customer) – Customer to use in instantiation request
- **owning_entity** (OwningEntity) – Owning entity to use in instantiation request
- **project** (Project) – Project name to use in instantiation request
- **line_of_business_object** (LineOfBusiness) – LineOfBusiness name to use in instantiation request
- **platform_object** (Platform) – Platform name to use in instantiation request
- **aai_service** (*AaiService*) – Service object from aai sdc
- **cloud_region** (CloudRegion) – Cloud region to use in instantiation request
- **tenant** (Tenant) – Tenant to use in instantiation request
- **service_instance_name** (*str, optional*) – Service instance name. Defaults to None.
- **vnf_parameters** (Iterable[ForwardRef]) – (Iterable[VnfParameters]): Parameters which are going to be used for vnfs instantiation. Defaults to None.
- **enable_multicloud** (*bool, optional*) – Determines if Multicloud should be enabled for instantiation request. Defaults to False.
- **so_service** (SoService, *optional*) – SO values to use in instantiation request
- **service_subscription** (ServiceSubscription, *optional*) – Customer service subscription for the instantiated service. Required if so_service is not provided.

Raises ***StatusError*** – if a service is not distributed.

Returns instantiation request object

Return type *ServiceInstantiation*

class onapsdk.so.instantiation.**SoService**(*subscription_service_type*, *vnfs=<factory>*, *pnfs=<factory>*, *instance_name=None*)

Bases: object

Class to store SO Service parameters used for macro instantiation.

Contains value list: List of vnfs to instantiate Contains value: subscription service type

instance_name: Optional[str] = None

classmethod load(*data*)

Create a service instance description object from the dict.

Useful if you keep your instance data in file.

Returns SoService object created from the dictionary

Return type *SoService*

pnfs: List[*onapsdk.so.instantiation.SoServicePnf*]

subscription_service_type: str

vnfs: List[*onapsdk.so.instantiation.SoServiceVnf*]

class onapsdk.so.instantiation.**SoServicePnf**(*model_name*, *instance_name*, *parameters=<factory>*, *processing_priority=None*)

Bases: *onapsdk.so.instantiation.SoServiceXnf*

Class to store a Pnf instance parameters.

instance_name: *str*

model_name: *str*

parameters: *Dict[str, Any]*

class onapsdk.so.instantiation.**SoServiceVfModule**(*model_name*, *instance_name*, *parameters=<factory>*, *processing_priority=None*)

Bases: object

Class to store a VfModule instance parameters.

instance_name: *str*

model_name: *str*

parameters: *Dict[str, Any]*

processing_priority: *Optional[int] = None*

class onapsdk.so.instantiation.**SoServiceVnf**(*model_name*, *instance_name*, *parameters=<factory>*, *processing_priority=None*, *vf_modules=<factory>*)

Bases: *onapsdk.so.instantiation.SoServiceXnf*

Class to store a Vnf instance parameters.

vf_modules: *List[**onapsdk.so.instantiation.SoServiceVfModule**]*

class onapsdk.so.instantiation.**SoServiceXnf**(*model_name*, *instance_name*, *parameters=<factory>*, *processing_priority=None*)

Bases: object

Class to store a Xnf instance parameters.

instance_name: *str*

classmethod load(*data*)

Create a vnf instance description object from the dict.

Useful if you keep your instance data in file.

Returns SoServiceVnf object created from the dictionary

Return type *SoServiceVnf*

model_name: *str*

parameters: *Dict[str, Any]*

processing_priority: *Optional[int] = None*

class onapsdk.so.instantiation.**Subnet**(*name*, *start_address*, *gateway_address*, *role=None*, *cidr_mask='24'*, *ip_version='4'*, *dhcp_enabled=False*, *dhcp_start_address=None*, *dhcp_end_address=None*)

Bases: object

Class to store subnet parameters used for preload.

cidr_mask: *str = '24'*

dhcp_enabled: *bool = False*

dhcp_end_address: *Optional[str] = None*

dhcp_start_address: *Optional[str] = None*

gateway_address: str

ip_version: str = '4'

name: str

role: str = None

start_address: str

class onapsdk.so.instantiation.**VfModuleInstantiation**(*name*, *request_id*, *instance_id*, *vf_module*)

Bases: *onapsdk.so.instantiation.Instantiation*

VF module instantiation class.

classmethod instantiate_ala_carte(*vf_module*, *vnf_instance*, *cloud_region*, *tenant*, *vf_module_instance_name=None*, *vnf_parameters=None*, *use_preload=True*)

Instantiate VF module.

Iterate throught vf modules from service Tosca file and instantiate vf modules.

Parameters

- **vf_module** (VfModule) – VfModule to instantiate
- **vnf_instance** (VnfInstance) – VnfInstance object
- **cloud_region** (CloudRegion, *optional*) – Cloud region to use in instantiation request. Defaults to None.
- **tenant** (Tenant, *optional*) – Tenant to use in instnatiation request. Defaults to None.
- **vf_module_instance_name_factory** (*str*, *optional*) – Factory to create VF module names. It's going to be a prefix of name. Index of vf module in Tosca file will be added to it. If no value is provided it's going to be "Python_ONAP_SDK_vf_module_service_instance_{str(uuid4())}". Defaults to None.
- **vnf_parameters** (*Iterable[InstantiationParameter]*, *optional*) – Parameters which are going to be used in preload upload for vf modules or passed in "userParams". Defaults to None.
- **use_preload** (*bool*, *optional*) – This flag determines whether instantiation parameters are used as preload or "userParams" content. Defaults to True

Yields *Iterator[VfModuleInstantiation]* – VfModuleInstantiation class object.

Return type *VfModuleInstantiation*

class onapsdk.so.instantiation.**VfmoduleParameters**(*name*, *vfmodule_parameters=None*)

Bases: object

Class to store vfmodule parameters used for macro instantiation.

Contains value lists: List of vfModule parameters

name: str

vfmodule_parameters: Iterable[*onapsdk.so.instantiation.InstantiationParameter*] = None

class onapsdk.so.instantiation.**VnfInstantiation**(*name*, *request_id*, *instance_id*, *line_of_business*, *platform*, *vnf*)

Bases: *onapsdk.so.instantiation.NodeTemplateInstantiation*

VNF instantiation class.

classmethod create_from_request_response(*request_response*)
Create VNF instantiation object based on request details.

Raises

- ***ResourceNotFound*** – Service related with given object doesn't exist
- ***ResourceNotFound*** – No ServiceInstantiation related with given VNF instantiation
- ***ResourceNotFound*** – VNF related with given object doesn't exist
- ***InvalidResponse*** – Invalid dictionary - couldn't create VnfInstantiation object

Returns VnfInstantiation object

Return type *VnfInstantiation*

classmethod get_by_vnf_instance_name(*vnf_instance_name*)
Get VNF instantiation request by instance name.

Raises ***InvalidResponse*** – Vnf instance with given name does not contain requestList or the requestList does not contain any details.

Returns Vnf instantiation request object

Return type *VnfInstantiation*

classmethod instantiate_ala_carte(*aai_service_instance*, *vnf_object*, *line_of_business*, *platform*, *cloud_region*, *tenant*, *sdc_service*, *vnf_instance_name=None*, *vnf_parameters=None*)
Instantiate Vnf using a'la carte method.

Parameters

- **vnf_object** (Vnf) – Vnf to instantiate
- **line_of_business_object** (LineOfBusiness) – LineOfBusiness to use in instantiation request
- **platform_object** (Platform) – Platform to use in instantiation request
- **cloud_region** (CloudRegion) – Cloud region to use in instantiation request.
- **tenant** (Tenant) – Tenant to use in instnatiation request.
- **vnf_instance_name** (*str, optional*) – Vnf instance name. Defaults to None.
- **vnf_parameters** (*Iterable[InstantiationParameter], optional*) – Instantiation parameters that are sent in the request. Defaults to None

Returns VnfInstantiation object

Return type *VnfInstantiation*

classmethod instantiate_macro(*aai_service_instance*, *vnf_object*, *line_of_business*, *platform*, *cloud_region*, *tenant*, *sdc_service*, *vnf_instance_name=None*, *vnf_parameters=None*, *so_vnf=None*)
Instantiate Vnf using macro method.

Parameters

- **aai_service_instance** (ServiceInstance) – Service instance associated with request
- **vnf_object** (Vnf) – Vnf to instantiate

- **line_of_business** (LineOfBusiness) – LineOfBusiness to use in instantiation request
- **platform** (Platform) – Platform to use in instantiation request
- **cloud_region** (CloudRegion) – Cloud region to use in instantiation request.
- **tenant** (Tenant) – Tenant to use in instantiation request.
- **vnf_instance_name** (*str, optional*) – Vnf instance name. Defaults to None.
- **vnf_parameters** (*Iterable[InstantiationParameter], optional*) – Instantiation parameters that are sent in the request. Defaults to None
- **so_vnf** (SoServiceVnf) – object with vnf instance parameters

Returns VnfInstantiation object

Return type *VnfInstantiation*

classmethod so_action(*vnf_instance*, *operation_type*, *aai_service_instance*, *line_of_business*, *platform*, *sdc_service*, *so_service=None*)

Execute SO action (update or healthcheck) for selected vnf with SO macro request.

Parameters

- **vnf_instance** (VnfInstance) – vnf instance object
- **operation_type** (VnfOperation) – name of the operation to trigger
- **aai_service_instance** (*AaiService*) – Service Instance object from aai
- **line_of_business** (LineOfBusiness) – LineOfBusiness name to use in instantiation request
- **platform** (Platform) – Platform name to use in instantiation request
- **sdc_service** (*SdcService*) – Service model information
- **so_service** (SoService, *optional*) – SO values to use in SO request

Raises ***StatusError*** – if the provided operation is not supported

Returns VnfInstantiation object

Return type *VnfInstantiation*

class onapsdk.so.instantiation.**VnfOperation**(*request_method*, *request_suffix*)

Bases: *onapsdk.so.instantiation.Operation*

Class to store possible operations' data for vnfs (request method and suffix).

HEALTHCHECK = Operation(request_method='POST', request_suffix='/healthcheck')

UPDATE = Operation(request_method='PUT', request_suffix='')

request_method: str

request_suffix: str

class onapsdk.so.instantiation.**VnfParameters**(*name*, *vnf_parameters=None*, *vfmodule_parameters=None*)

Bases: object

Class to store vnf parameters used for macro instantiation.

Contains value lists: List vnf Instantiation parameters and list of vfModule parameters

name: str

vfmodule_parameters: Iterable[*onapsdk.so.instantiation.VfmoduleParameters*] = None

vnf_parameters: Iterable[*onapsdk.so.instantiation.InstantiationParameter*] = None

onapsdk.so.so_db_adapter module

Database Adapter module.

class onapsdk.so.so_db_adapter.**IdentityService**(*identity_id*, *url='http://1.2.3.4:5000/v2.0'*, *mso_id='onapsdk_user'*, *mso_pass='mso_pass_onapsdk'*, *project_domain_name='NULL'*, *user_domain_name='NULL'*, *admin_tenant='service'*, *member_role='admin'*, *identity_server_type='KEYSTONE'*, *identity_authentication_type='USERNAME_PASSWORD'*, *server_type_as_string='KEYSTONE'*, *tenant_metadata=True*)

Bases: object

Class to store identity service details.

admin_tenant: str = 'service'

hibernate_lazy_initializer = {}

identity_authentication_type: str = 'USERNAME_PASSWORD'

identity_id: str

identity_server_type: str = 'KEYSTONE'

member_role: str = 'admin'

mso_id: str = 'onapsdk_user'

mso_pass: str = 'mso_pass_onapsdk'

project_domain_name: str = 'NULL'

server_type_as_string: str = 'KEYSTONE'

tenant_metadata: bool = True

url: str = 'http://1.2.3.4:5000/v2.0'

user_domain_name: str = 'NULL'

class onapsdk.so.so_db_adapter.**SoDbAdapter**(*name=None*, *_server='SO'*, *_status=None*)

Bases: *onapsdk.so.so_element.SoElement*, abc.ABC

DB Adapter class.

classmethod add_cloud_site(*cloud_region_id*, *complex_id*, *identity_service*, *orchestrator='multicloud'*)

Add cloud_site data with identity_service to SO db.

Parameters

- **cloud_region_id** (*str*) – The id of cloud region
- **complex_id** (*str*) – The id of complex

- **identity_service** (IdentityService) – Identity service related to the cloud region
- **orchestrator** (*str, optional*) – Orchestrator type. Defaults to multicloud.

Important: identity_services data will be overwrite, but in the same time cloud_sites data will not (shouldn't) be overwrite! SOCatalogDB REST API has some limitations reported: https://jira.onap.org/browse/SO-2727

Returns response object

classmethod get_service_vnf_info(*identifier*)

Get Service VNF and VF details.

Return type Dict[Any, Any]

Returns The response in a dict format

onapsdk.so.so_element module

SO Element module.

class onapsdk.so.so_element.**OrchestrationRequest**(*request_id*)

Bases: *onapsdk.so.so_element.SoElement*, *onapsdk.utils.mixins.WaitForFinishMixin*, abc.ABC

Base SO orchestration request class.

class StatusEnum(*value*)

Bases: enum.Enum

Status enum.

Store possible statuses for instantiation:

- IN_PROGRESS,
- FAILED,
- COMPLETE.

If instantiation has status which is not covered by these values UNKNOWN value is used.

COMPLETED = 'COMPLETE'

FAILED = 'FAILED'

IN_PROGRESS = 'IN_PROGRESS'

UNKNOWN = 'UNKNOWN'

WAIT_FOR_SLEEP_TIME = 10

property completed

Store an information if instantion is completed or not.

Instantiation is completed if it's status is COMPLETED.

Returns True if instantiation is completed, False otherwise.

Return type bool

property failed
Store an information if instantion is failed or not.

Instantiation is failed if it's status is FAILED.

Returns True if instantiation is failed, False otherwise.

Return type bool

property finished
Store an information if instantion is finished or not.

Instantiation is finished if it's status is COMPLETED or FAILED.

Returns True if instantiation is finished, False otherwise.

Return type bool

property status
Object instantiation status.

It's populated by call SO orchestation request endpoint.

Returns Instantiation status.

Return type *StatusEnum*

class onapsdk.so.so_element.**SoElement**(*name=None*, *_server='SO'*, *_status=None*)
Bases: *onapsdk.onap_service.OnapService*

Mother Class of all SO elements.

api_version = 'v7'

base_url = 'http://so.api.simpledemo.onap.org:30277'

classmethod get_guis()
Retrieve the status of the SO GUIs.

Only one GUI is referenced for SO: SO monitor

Return the list of GUIs

Return type GuiItem

classmethod get_service_model_info(*service_name*)
Retrieve Service Model info.

classmethod get_subscription_service_type(*vf_name*)
Retrieve the model info of the VFs.

classmethod get_vf_model_info(*vf_model*)
Retrieve the VF model info From Tosca?.

Return type str

classmethod get_vnf_model_info(*vf_name*)
Retrieve the model info of the VFs.

property headers
Create headers for SO request.

It is used as a property because x-transactionid header should be unique for each request.

name: str = None

Module contents

ONAP SDK SO package.

onapsdk.utils package

Submodules

onapsdk.utils.configuration module

Configuration package.

onapsdk.utils.configuration.**components_needing_distribution**()
Return the list of components needing distribution.

Return type List[str]

onapsdk.utils.configuration.**tosca_path**()
Return tosca file paths.

Return type str

onapsdk.utils.headers_creator module

Header creator package.

onapsdk.utils.headers_creator.**headers_aai_creator**(*base_header*)
Create the right headers for AAI creator type.

Parameters **base_header** (*Dict[str, str]*) – the base header to use

Returns the needed headers

Return type Dict[str, str]

onapsdk.utils.headers_creator.**headers_clamp_creator**(*base_header*)
Create the right headers for CLAMP generic type.

base_header (Dict[str, str]): the base header to use data (str): payload data used to create an md5 content header

Returns the needed headers

Return type Dict[str, str]

onapsdk.utils.headers_creator.**headers_msb_creator**(*base_header*)
Create the right headers for MSB.

Parameters **base_header** (*Dict[str, str]*) – the base header to use

Returns the needed headers

Return type Dict[str, str]

onapsdk.utils.headers_creator.**headers_sdc_artifact_upload**(*base_header*, *data*)
Create the right headers for sdc artifact upload.

Parameters

- **base_header** (*Dict[str, str]*) – the base header to use

- **data** (*str*) – payload data used to create an md5 content header

Returns the needed headers

Return type Dict[str, str]

onapsdk.utils.headers_creator.**headers_sdc_creator**(*base_header*, *user='cs0008'*, *authorization=None*)

Create the right headers for SDC creator type.

Parameters

- **base_header** (*Dict[str, str]*) – the base header to use
- **user** (*str, optional*) – the user to use. Default to cs0008
- **authorization** (*str, optional*) – the basic auth to use. Default to "classic" one

Returns the needed headers

Return type Dict[str, str]

onapsdk.utils.headers_creator.**headers_sdc_generic**(*base_header*, *user*, *authorization=None*)

Create the right headers for SDC generic type.

Parameters

- **base_header** (*Dict[str, str]*) – the base header to use
- **user** (*str*) – the user to use.
- **authorization** (*str, optional*) – the basic auth to use. Default to "classic" one

Returns the needed headers

Return type Dict[str, str]

onapsdk.utils.headers_creator.**headers_sdc_governor**(*base_header*, *user='gv0001'*, *authorization=None*)

Create the right headers for SDC governor type.

Parameters

- **base_header** (*Dict[str, str]*) – the base header to use
- **user** (*str, optional*) – the user to use. Default to gv0001
- **authorization** (*str, optional*) – the basic auth to use. Default to "classic" one

Returns the needed headers

Return type Dict[str, str]

onapsdk.utils.headers_creator.**headers_sdc_operator**(*base_header*, *user='op0001'*, *authorization=None*)

Create the right headers for SDC operator type.

Parameters

- **base_header** (*Dict[str, str]*) – the base header to use
- **user** (*str, optional*) – the user to use. Default to op0001
- **authorization** (*str, optional*) – the basic auth to use. Default to "classic" one

Returns the needed headers

Return type Dict[str, str]

onapsdk.utils.headers_creator.**headers_sdc_tester**(*base_header*, *user='jm0007'*, *authorization=None*)

Create the right headers for SDC tester type.

Parameters

- **base_header** (*Dict[str, str]*) – the base header to use
- **user** (*str, optional*) – the user to use. Default to jm0007
- **authorization** (*str, optional*) – the basic auth to use. Default to "classic" one

Returns the needed headers

Return type Dict[str, str]

onapsdk.utils.headers_creator.**headers_sdnc_creator**(*base_header*)

Create the right headers for SDNC.

Parameters **base_header** (*Dict[str, str]*) – the base header to use

Returns the needed headers

Return type Dict[str, str]

onapsdk.utils.headers_creator.**headers_so_catelog_db_creator**(*base_header*)

Create the right headers for SO creator type.

Parameters **base_header** (*Dict[str, str]*) – the base header to use

Returns the needed headers

Return type Dict[str, str]

onapsdk.utils.headers_creator.**headers_so_creator**(*base_header*)

Create the right headers for SO creator type.

Parameters **base_header** (*Dict[str, str]*) – the base header to use

Returns the needed headers

Return type Dict[str, str]

onapsdk.utils.jinja module

Jinja module.

onapsdk.utils.jinja.**jinja_env**()

Create Jinja environment.

jinja_env allow to fetch simply jinja templates where they are. by default jinja engine will look for templates in *templates* directory of the package. So to load a template, you just have to do:

Example: >>> template = jinja_env().get_template('vendor_create.json.j2') >>> data = template.render(name="vendor")

See also:

SdcElement.create() for real use

Returns the Jinja environment to use

Return type Environment

onapsdk.utils.mixins module

Mixins module.

class onapsdk.utils.mixins.**WaitForFinishMixin**
Bases: abc.ABC

Wait for finish mixin.

Mixin with wait_for_finish method and two properties:

- completed,
- finished.

Can be used to wait for result of asynchronous tasks.

WAIT_FOR_SLEEP_TIME = 10

abstract property completed
Store an information if object task is completed or not.

Returns True if object task is completed, False otherwise.

Return type bool

abstract property finished
Store an information if object task is finished or not.

Returns True if object task is finished, False otherwise.

Return type bool

wait_for_finish(*timeout=None*)
Wait until object task is finished.

It uses time.sleep with WAIT_FOR_SLEEP_TIME value as a parameter to wait unitl request is finished (object's finished property is equal to True).

It runs another process to control time of the function. If process timed out TimeoutError is going to be raised.

Parameters **timeout** (*float, optional*) – positive number, wait at most timeout seconds

Raises **TimeoutError** – Raised when function timed out

Returns True if object's task is successfully completed, False otherwise

Return type bool

onapsdk.utils.tosca_file_handler module

Utils class.

onapsdk.utils.tosca_file_handler.**get_modules_list_from_tosca_file**(*model*)
Get the list of modules from tosca file.

Modules are stored on topology_template.groups TOSCA file section.

Parameters **model** (*str*) – the model retrieved from the tosca file at Vnf instantiation.

Returns a list of modules

Return type dict

onapsdk.utils.tosca_file_handler.**get_parameter_from_yaml**(*parameter*, *config_file*)
Get the value of a given parameter in file.yaml.

Parameter must be given in string format with dots Example: general.openstack.image_name

Parameters

- **parameter** (*str*) –
- **config_file** (*str*) – configuration yaml file formtatted as string

Raises ***ParameterError*** – parameter not defined

Returns the value of the parameter

onapsdk.utils.tosca_file_handler.**get_vf_list_from_tosca_file**(*model*)
Get the list of Vfs of a VNF based on the tosca file.

Parameters **model** (*str*) – the model retrieved from the tosca file at Vnf instantiation

Returns a list of Vfs

Return type list

onapsdk.utils.tosca_file_handler.**random_string_generator**(*size=6*, *chars='ABCDEFGHIJKLMNOPQRSTUVWXYZ0*
Get a random String for VNF.

Parameters

- **size** (*int*) – the number of alphanumerical chars for CI
- **chars** (*str*) – alphanumerical characters (ASCII uppercase and digits)

Returns a sequence of random characters

Return type str

Module contents

ONAP SDK utils package.

onapsdk.utils.**get_zulu_time_isoformat**()
Get zulu time in accepted by ONAP modules format.

Returns Actual Zulu time.

Return type str

onapsdk.utils.**load_json_file**(*path_to_json_file*)
Return json as string from selected file.

Parameters **path_to_json_file** (*str*) – (str) path to file with json

Return type str

Returns File content as string (str)

onapsdk.ves package

Submodules

onapsdk.ves.ves module

Base VES event sender.

class onapsdk.ves.ves.**Ves**
Bases: *onapsdk.ves.ves_service.VesService*

Ves library provides functions for sending events to VES.

event_batch_endpoint_url: str = '{}/eventListener/{}/eventBatch'

event_endpoint_url: str = '{}/eventListener/{}'

classmethod send_batch_event(*version*, *json_event*, *basic_auth*)
Send a batch event stored in a file to VES.

Parameters

- **version** (str) – (str) version of VES data format
- **json_event** (str) – (str) event to send
- **basic_auth** (Dict[str, str]) – Dict[str, str], for example:{ 'username': 'bob', 'password': 'secret' }

Return type Optional[Response]

Returns (requests.Response) HTTP response status

classmethod send_event(*version*, *json_event*, *basic_auth*)
Send an event stored in a file to VES.

Parameters

- **version** (str) – (str) version of VES data format
- **json_event** (str) – (str) event to send
- **basic_auth** (Dict[str, str]) – Dict[str, str], for example:{ 'username': 'bob', 'password': 'secret' }

Return type Optional[Response]

Returns (requests.Response) HTTP response status

onapsdk.ves.ves_service module

Base VES module.

class onapsdk.ves.ves_service.**VesService**
Bases: *onapsdk.onap_service.OnapService*

Base VES class.

Stores url to VES API (edit if you want to use other) and authentication tuple (username, password).

Module contents

ONAP SDK VES package.

onapsdk.vid package

Submodules

onapsdk.vid.vid module

VID module.

class onapsdk.vid.vid.**LineOfBusiness**(*name*)
Bases: *onapsdk.vid.vid.Vid*

VID line of business class.

classmethod get_create_url()
Line of business creation url.

Returns Url used for line of business creation

Return type str

class onapsdk.vid.vid.**OwningEntity**(*name*)
Bases: *onapsdk.vid.vid.Vid*

VID owning entity class.

classmethod get_create_url()
Owning entity creation url.

Returns Url used for ownint entity creation

Return type str

class onapsdk.vid.vid.**Platform**(*name*)
Bases: *onapsdk.vid.vid.Vid*

VID platform class.

classmethod get_create_url()
Platform creation url.

Returns Url used for platform creation

Return type str

class onapsdk.vid.vid.**Project**(*name*)
Bases: *onapsdk.vid.vid.Vid*

VID project class.

classmethod get_create_url()
Project creation url.

Returns Url used for project creation

Return type str

class onapsdk.vid.vid.**Vid**(*name*)
Bases: *onapsdk.onap_service.OnapService*, abc.ABC

VID base class.

api_version = '/vid'

base_url = 'https://vid.api.simpledemo.onap.org:30200'

classmethod create(*name*)
Create VID resource.

Returns Created VID resource

Return type *Vid*

classmethod get_create_url()
Resource url.

Used to create resources

Returns Url used for resource creation

Return type str

Module contents

VID package.

8.1.2 Submodules

8.1.3 onapsdk.constants module

Constant package.

8.1.4 onapsdk.exceptions module

ONAP Exception module.

exception onapsdk.exceptions.**APIError**(*message=None*, *response_status_code=None*)
Bases: *onapsdk.exceptions.RequestError*

API error occured.

property response_status_code
Response status code property.

Returns Response status code. If not set, returns 0

Return type int

exception onapsdk.exceptions.**ConnectionFailed**
Bases: *onapsdk.exceptions.RequestError*

Unable to connect.

exception onapsdk.exceptions.**FileError**
Bases: *onapsdk.exceptions.ValidationError*

Reading in a file failed.

exception onapsdk.exceptions.**InvalidResponse**
Bases: *onapsdk.exceptions.RequestError*

Unable to decode response.

exception onapsdk.exceptions.**ModuleError**
Bases: *onapsdk.exceptions.SDKException*

Unable to import module.

exception onapsdk.exceptions.**NoGuiError**
Bases: *onapsdk.exceptions.SDKException*

No GUI available for this component.

exception onapsdk.exceptions.**ParameterError**
Bases: *onapsdk.exceptions.SDKException*

Parameter does not satisfy requirements.

exception onapsdk.exceptions.**RelationshipNotFound**(*message=None*, *response_status_code=None*)
Bases: *onapsdk.exceptions.ResourceNotFound*

Required relationship is missing.

exception onapsdk.exceptions.**RequestError**
Bases: *onapsdk.exceptions.SDKException*

Request error occured.

exception onapsdk.exceptions.**ResourceNotFound**(*message=None*, *response_status_code=None*)
Bases: *onapsdk.exceptions.APIError*

Requested resource does not exist.

exception onapsdk.exceptions.**SDKException**
Bases: Exception

Generic exception for ONAP SDK.

exception onapsdk.exceptions.**SettingsError**
Bases: *onapsdk.exceptions.SDKException*

Some settings are wrong.

exception onapsdk.exceptions.**StatusError**
Bases: *onapsdk.exceptions.SDKException*

Invalid status.

exception onapsdk.exceptions.**ValidationError**
Bases: *onapsdk.exceptions.SDKException*

Data validation failed.

8.1.5 onapsdk.onap_service module

ONAP Service module.

class onapsdk.onap_service.**OnapService**
Bases: abc.ABC

Mother Class of all ONAP services.

An important attribute when inheriting from this class is *_jinja_env*. it allows to fetch simply jinja templates where they are. by default jinja engine will look for templates in *templates* directory of the package. See in Examples to see how to use.

server
nickname of the server we send the request. Used in logs strings. For example, 'SDC' is the nickame for SDC server.

Type str

headers
the headers dictionnary to use.

Type Dict[str, str]

proxy
the proxy configuration if needed.

Type Dict[str, str]

permanent_headers
optional dictionary of headers which could be set by the user and which are **always** added into sended request. Unlike the *headers*, which could be overrided on *send_message* call these headers are constant.

Type Optional[Dict[str, str]]

class PermanentHeadersCollection(*ph_dict=<factory>*, *ph_call=<factory>*)
Bases: object

Collection to store permanent headers.

ph_call: List[Callable]

ph_dict: Dict[str, Any]

classmethod get_guis()
Return the list of GUI and its status.

headers: Dict[str, str] = {'Accept': 'application/json', 'Content-Type': 'applicati

permanent_headers: *onapsdk.onap_service.OnapService.PermanentHeadersCollection* = Onap

proxy: Dict[str, str] = None

classmethod send_message(*method*, *action*, *url*, ***kwargs*)
Send a message to an ONAP service.

Parameters

- **method** (*str*) – which method to use (GET, POST, PUT, PATCH, . . .)
- **action** (*str*) – what action are we doing, used in logs strings.
- **url** (*str*) – the url to use
- **exception** (*Exception, optional*) – if an error occurs, raise the exception given instead of RequestError

- ****kwargs** – Arbitrary keyword arguments. any arguments used by requests can be used here.

Raises

- ***RequestError*** – if other exceptions weren't caught or didn't raise, or if there was an ambiguous exception by a request
- ***ResourceNotFound*** – 404 returned
- ***APIError*** – returned an error code within 400 and 599, except 404
- ***ConnectionFailed*** – connection can't be established

Return type Optional[Response]

Returns the request response if OK

classmethod send_message_json(*method*, *action*, *url*, ***kwargs*)

Send a message to an ONAP service and parse the response as JSON.

Parameters

- **method** (*str*) – which method to use (GET, POST, PUT, PATCH, ...)
- **action** (*str*) – what action are we doing, used in logs strings.
- **url** (*str*) – the url to use
- **exception** (*Exception, optional*) – if an error occurs, raise the exception given
- ****kwargs** – Arbitrary keyword arguments. any arguments used by requests can be used here.

Raises

- ***InvalidResponse*** – if JSON coudn't be decoded
- ***RequestError*** – if other exceptions weren't caught or didn't raise
- **APIError/ResourceNotFound** – send_message() got an HTTP error code
- ***ConnectionFailed*** – connection can't be established
- ***RequestError*** – send_message() raised an ambiguous exception

Return type Dict[Any, Any]

Returns the response body in dict format if OK

server: str = None

static set_header(*header=None*)

Set the header which will be always send on request.

The header can be:

- dictionary - will be used same dictionary for each request
- callable - a method which is going to be called every time on request creation. Could be useful if you need to connect with ONAP through some API gateway and you need to take care about authentication. The callable shouldn't require any parameters
- None - reset headers

Parameters **header** (*Optional[Union[Dict[str, Any], Callable]]*) – header to set. Defaults to None

Return type None

static set_proxy(*proxy*)
Set the proxy for Onap Services rest calls.

Parameters **proxy** (*Dict[str, str]*) – the proxy configuration

Examples

```
>>> OnapService.set_proxy({
...     'http': 'socks5h://127.0.0.1:8082',
...     'https': 'socks5h://127.0.0.1:8082'})
```

Return type None

8.1.6 onapsdk.version module

Version module.

8.1.7 Module contents

ONAP SDK master package.

CHAPTER

NINE

DESCRIPTION

ONAP SDK is a client library written in Python for building applications to work with ONAP. The project aims to provide a consistent and complete set of interactions with ONAP's many services, along with complete documentation, examples, and tools.

Using few python commands, you should be able to onboard, distribute models and instantiate xNFs.

First beta release deals with ONAP "Legacy" APIs but new APIs, CDS and policy integration is planned for next releases.

CHAPTER

TEN

INDICES AND TABLES

- modindex
- glossary

PYTHON MODULE INDEX

O

INDEX

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W