CODE:

#Application SDK wiotp.sdk.application.ApplicationClient

#Configuration

myConfig = {

"key": "a-org1id-y67si9et"

"token": "Ab$76s)asj8\_s5"

}

}

client = wiotp.sdk.application.ApplicationClient(config=myConfig)

#Application Configuration

myConfig = {

"identity": {

"appId": "app1"

}.

"auth" {

"key": "orgid-h798S783DK"

"token": "Ab$76s)asj8\_s5"

},

"options": {

"domain": "internetofthings.ibmcloud.com",

"logLevel": "error|warning|info|debug",

"http": {

"verify": True|False

},

"mqtt": {

"instanceId": "instance1",

"port": 8883,

"transport": "tcp|websockets",

"cleanStart": True|False,

"sessionExpiry": 3600,

"keepAlive": 60,

"caFile": "/path/to/certificateAuthorityFile.pem"

}

}

}

client = wiotp.sdk.application.ApplicationClient(config=myConfig, logHandlers=None)

#YAML File Support

import wiotp.sdk.application

myConfig = wiotp.sdk.application.parseConfigFile("application.yaml")

client = wiotp.sdk.application.ApplicationClient(config=myConfig, logHandlers=None)

#Publishing Commands to Devices

import wiotp.sdk.application

options = wiotp.sdk.application.ParseConfigFile("app.yaml")

client = wiotp.sdk.application.ApplicationClient(options)

client.connect()

commandData={'rebootDelay' : 50}

client.publishCommand(myDeviceType, myDeviceId, "reboot", "json", commandData)

#Subscribing to Device Events

import wiotp.sdk.application

options = wiotp.sdk.application.parseConfigFile("app.yaml")

client = wiotp.sdk.application.ApplicationClient(options)

client.connect()

# Subscribing to all events from all devices

client.subscribeToDeviceEvents()

# Subscribing to all events from all devices of a specific type

client.subscribeToDeviceEvents(typeId=myDeviceType)

# Subscribing to a specific event from all devices

client.subscribeToDeviceEvents(eventId=myEvent)

# Subscribing to a specific event from two or more different devices

client.subscribeToDeviceEvents(typeId=myDeviceType, deviceId=myDeviceId, eventId=myEvent)

client.subscribeToDeviceEvents(typeId=myOtherDeviceType, eventId=myEvent)

# Subscribing to all events that are published in JSON format

client.subscribeToDeviceEvents(msgFormat="json")

#Service Bindings

import wiotp.sdk.application

options = wiotp.sdk.application.parseEnvVars()

appClient = wiotp.sdk.application.ApplicationClient(options)

serviceBinding = {

"name": "test-cloudant",

"description": "Test Cloudant instance",

"type": "cloudant",

"credentials": {

"host": "hostname",

"port": 443,

"username": "username",

"password": "password

}

}

cloudantService = appClient.serviceBindings.create(serviceBinding)

serviceBinding = {

"name": "test-eventstreams",

"description": "Test EventStreams instance",

"type": "eventstreams",

"credentials": {

"api\_key": "myapikey",

"user": "myusername,

"password": "mypassword",

"kafka\_admin\_url": "myurl",

"kafka\_brokers\_sasl": [ "broker1", "broker2", "broker3", "broker4", "broker5" ]

}

}

eventstreamsService = appClient.serviceBindings.create(serviceBinding)

#Cloudant Connector

import wiotp.sdk.application

options = wiotp.sdk.application.parseEnvVars()

appClient = wiotp.sdk.application.ApplicationClient(options)

serviceBinding = {

"name": "test-cloudant",

"description": "Test Cloudant instance",

"type": "cloudant",

"credentials": {

"host": "hostname",

"port": 443,

"username": "username",

"password": "password"

}

}

cloudantService = appClient.serviceBindings.create(serviceBinding)

# Create the connector

connector = self.appClient.dsc.create(

name="connector1", type="cloudant", serviceId=cloudantService.id, timezone="UTC",

description="A test connector", enabled=True

)

# Create a destination under the connector

destination1 = connector.destinations.create(name="all-data", bucketInterval="DAY")

# Create a rule under the connector, that routes all events to the destination

rule1 = connector.rules.createEventRule(

name="allevents", destinationName=destination1.name, typeId="\*", eventId="\*",

description="Send all events", enabled=True

)

# Create a second rule under the connector, that routes all state to the same destination

rule2 = connector.rules.createStateRule(

name="allstate", destinationName=destination1.name, logicalInterfaceId="\*",

description="Send all state", enabled=True,

)