Introduction to DNA Center Report APIs

How to generate Client Detail report using dnacentersdk library

Juma Al Luwati
Customer Success Specialist

Contents

1.0 Why This Mini-Documentation:

It all started with a customer requesting to have a fully automated way of getting health score report of clients connecting to a particular SSID, so I started looking around for a way to achieve this. I came across a library called dnacentersdk, which simplifies the interaction and usage of DNA Center APIs, through which I could easily work with reports and eventually have that health score, but unfortunately, it wasn't easy nor well documented by the time this documentation was written, so I've made a lot of testing to finally get to the root of it all and thought why not to document my findings and share it with other API geeks out there.

The documentation starts by introducing the reader to the dnacentersdk library or package, then how to install it and interact with it, and how generate a client detail report. I've also included a link to GitHub repo that contains the full ready-to-use script that is mentioned in this documentation.

Hope you enjoy it and find it helpful.

2.0 Introduction to dnacentersdk Package:

2.1 What is dnacentersdk package and its benefits

"Simple, lightweight, scalable Python API wrapper for the DNA Center APIs"

The dnacentersdk package represents all of the Cisco DNA Center API interactions via native Python tools. Making working with the Cisco DNA Center APIs in Python a native and natural experience.

There are ways to interact with DNA Center APIs, on method is to uses the traditional ways of performing an API call (DNA Center APIs), using token, URLs, variables... etc. and the second method will utilize the dnacentersdk package. Let us use the two API methods to obtain a sample data from DNA Center in the next example and observe the difference.

Example: we will see an example of getting all the Enterprise SSIDs that exist on DNA Center, first using DNA Center APIs and secondly using dnacentersdk package to make the call.

2.1.1 Using DNA Center native APIs to extract all the SSID names on DNA Center

a. Prepare the environment and get the token

```
#GET THE TOKEN USING THIS CODE:

import requests
from requests.auth import HTTPBasicAuth
import json

requests.packages.urllib3.disable_warnings()
tokenurl = "https://{DNA CENTER URL}:/dna/system/api/v1/auth/token"

response = requests.request
(
    "POST",
    tokenurl,
    auth = HTTPBasicAuth("username", "password"), DNA CENTER CREDENTIALS
    verify = False
)
token = "" #TOKEN PLACEHOLDER

tokenDict = response.json()
for key,value in tokenDict.items():
  token += value
```

b. Make the API call

```
#Making the API call

url = "https://{DNA CENTER URL}/dna/intent/api/v1/enterprise-ssid" #API URL

headers = {"x-auth-token": token} #USING THE TOKEN FROM ABOVE
response = requests.request("GET", url, headers=headers, verify=False)

data = response.json()

counter = 1

```

```
for i in data:
  for x in i['ssidDetails']:

      print(str(counter)+':')
      print(x['name']) #THIS WILL PRINT THE SSID NAMES
      print(x['securityLevel']) #ADDING SECURITY LEVEL IN THE OUTPUT
      print(x['radioPolicy']) #ADDING RADIO BANDS IN THE OUTPUT
      print('----------------------') #ADDING HYPHENS TO SEPARATE THE SSIDS
      counter += 1

```

c. Observe the output from the above API call

```
$ python DNAC_API.py

1:
CHPG-BACKUP
wpa2_personal
Dual band operation with band select
----------------------
2:
SSID-DNAC_PSK
wpa2_personal
Dual band operation (2.4GHz and 5GHz)
----------------------
3:
NWide
wpa2_enterprise
5GHz only
----------------------
4:
NCC WIFI
wpa2_personal
Dual band operation (2.4GHz and 5GHz)
----------------------
5:
BTS-fabric-SSID
wpa2_enterprise
Dual band operation (2.4GHz and 5GHz)
----------------------
6:
WWS300
wpa2_enterprise
Dual band operation (2.4GHz and 5GHz)
----------------------
```

Basically, the script loops through the JSON output and prints the SSIDs names, security levels and bands in GHz. If we don't use loops; and just print the response, everything related to enterprise SSIDs will be returned including: clientExclusionTimeout, trafficType, authServer and more.

Let us go through a similar script using dnacentersdk package.

2.1.2 Using dnacentersdk package to extract all the SSID names on DNA Center

a. Add the credentials to access DNA Center at the top, and Make the API call

```
import urllib3
from dnacentersdk import DNACenterAPI
import json

urllib3.disable_warnings()
#THIS SECTION IS FOR CREDENTIALS
dnac = DNACenterAPI(username = 'username',
                    password = 'password',
                    base_url = 'https://{DNA CENTER URL}:443',
                    version  = '2.2.3.3', #THIS IS THE PACKAGE'S VERSION
                    verify = False)

data = dnac.wireless.get_enterprise_ssid() #THIS LINE OF CODE DOES THE WHOLE JOB

#LOOPING THROUGH THE OUTPUT TO OBTAIN SSID NAMES
counter = 1

for i in data:
    for x in i['ssidDetails']:

        print(str(counter)+':')
        print(x['name'])
        print(x['securityLevel'])
        print(x['radioPolicy'])
        print('----------------------')
        counter += 1
```

And the output when the code is executed is exactly the same as the previous output.

With the dnacentersdk package, we saved around 10 lines of code plus we did not use a token, which means that our coding environment in this case is always ready for API calls, no need to prepare it by obtaining a token from DNA Center or add any URLs to access the API.

This was a simple example however the package has many other API calls grouped according to a type such as Clients, Command Runner, Devices, SDA and more.

dnacentersdk helps you get things done faster. It takes care of the API semantics, and you can focus on writing your code.

2.2 How to download and use the dnacentersdk package

2.2.1 Downloading the package

dnacentersdk is available via PIP and the Python Package Index (PyPI). To install dnacentersdk, simply run this command from your terminal of choice:

```
$ pip install dnacentersdk
```

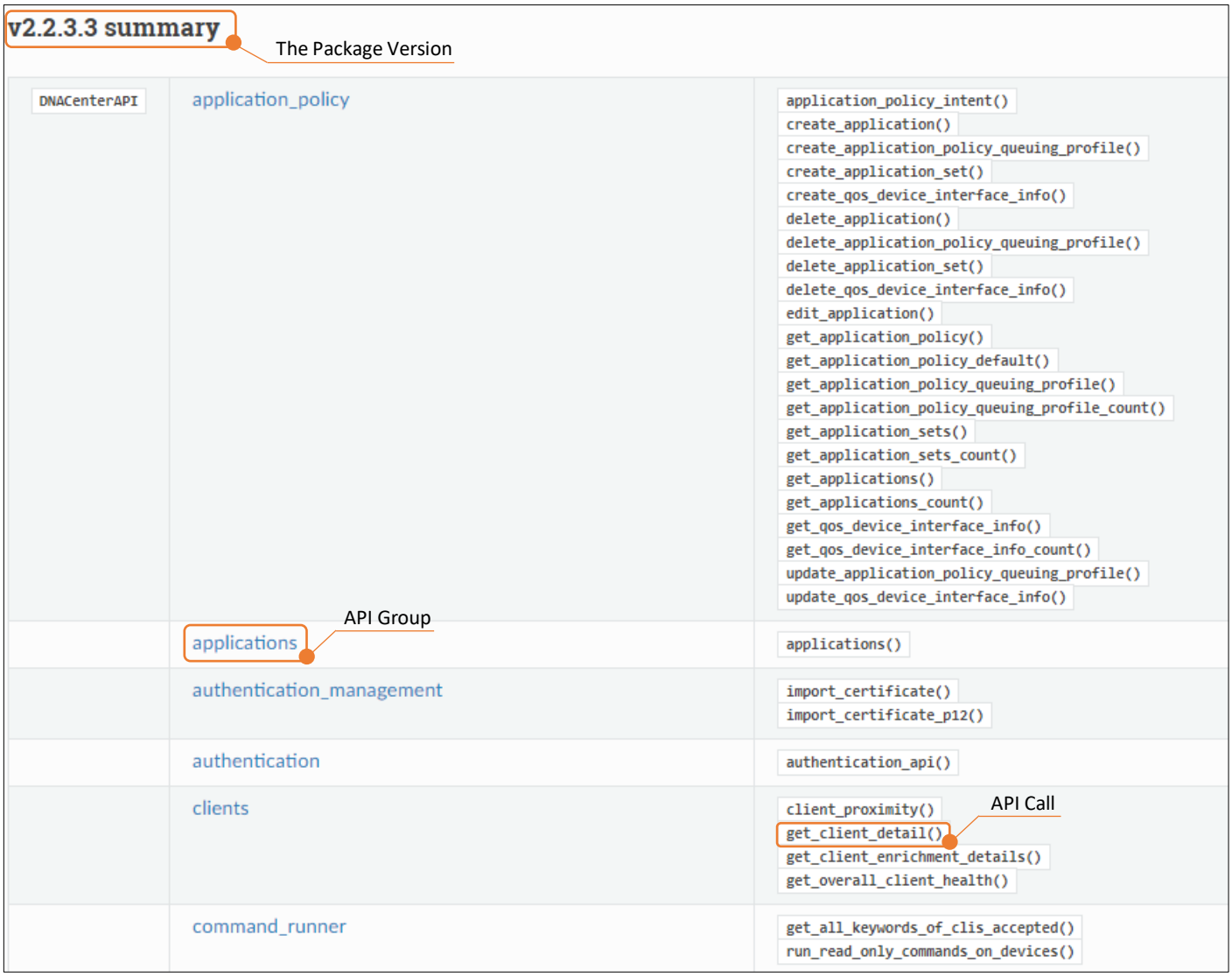
To upgrade to the latest version:

```
$ pip install dnacentersdk --upgrade
```

Once you have the package downloaded; you are ready to start exploring the dnacentersdk.

2.2.2 using dnacentersdk

If you go to the documentation of this package found on this link: https://dnacentersdk.readthedocs.io/ you will notice the following:



If you would like to make an API call, you need to check the dnacentersdk documentation, browse through the available APIs, click on a certain API group name, and finally read the requirements and make the call.

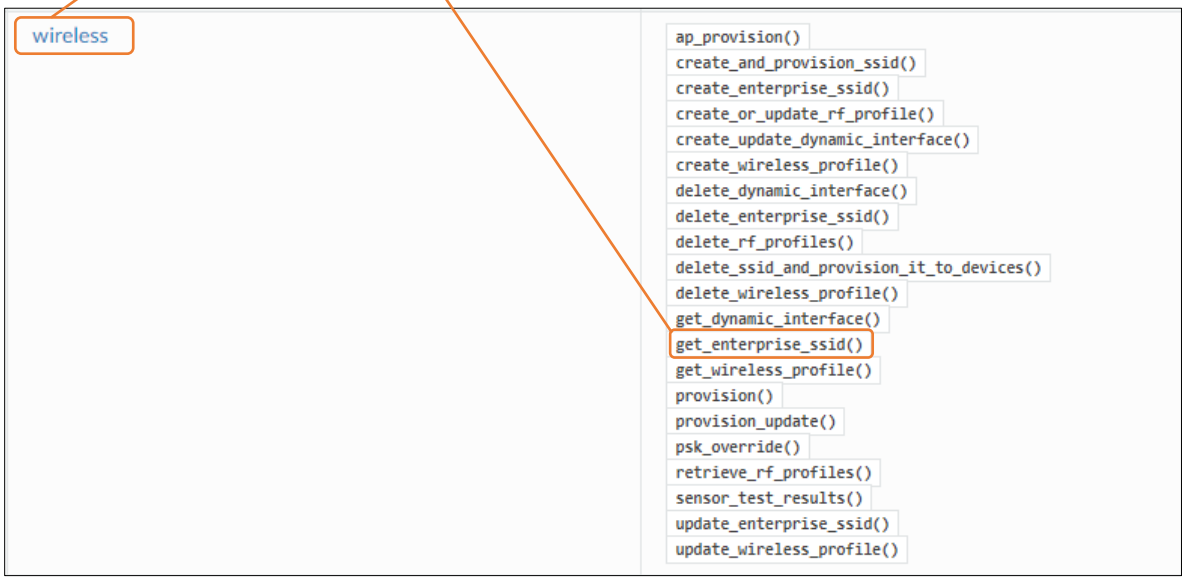
Every API call using dnacentersdk will have three components:

a. The word **"dnac"**
b. API group such as **"clients"**
c. API call such as **"get_client_detail( )"**

Note: the parenthesis contain the call parameters, which might or might not be needed depending on the API call and the REST method i.e. GET, POST, DELETE etc.

If we check again the previous example; where we've written a script to get all enterprise SSIDs from DNA Center, we will notice the main components of the API call as shown below:

```
1
2  data = dnac.wireless.get_enterprise_ssid() #THIS LINE OF CODE DOES THE WHOLE JOB
3
```



This particular API doesn't require any parameters on demand, however you could add "ssid_name" parameter to have data about one SSID, otherwise everything regarding all SSIDs will be presented

get_enterprise_ssid(*ssid_name=None, headers=None, **request_parameters*) [source]

Gets either one or all the enterprise SSID .

Parameters:
- **ssid_name** (*basestring*) – ssidName query parameter. Enter the enterprise SSID name that needs to be retrieved. If not entered, all the enterprise SSIDs will be retrieved.
- **headers** (*dict*) – Dictionary of HTTP Headers to send with the Request .
- ****request_parameters** – Additional request parameters (provides support for parameters that may be added in the future).

Returns: JSON response. A list of MyDict objects. Access the object's properties by using the dot notation or the bracket notation.

Return type: list

Raises:
- TypeError – If the parameter types are incorrect.
- MalformedRequest – If the request body created is invalid.
- ApiError – If the DNA Center cloud returns an error.

Again this was a simple GET API call, which usually contains no body and no parameters, however, the overall idea is the same for other APIs available in dnacentersdk package. In the next section, let's discuss DNA Center Reports, and their APIs.

3.0 DNA Center Client Details Report API:

3.1 Introduction

There's this item on DNA Center's menu called "Reports" which contains many report templates that could be generated on DNA Center's GUI, and then downloaded or utilized for various proposes.

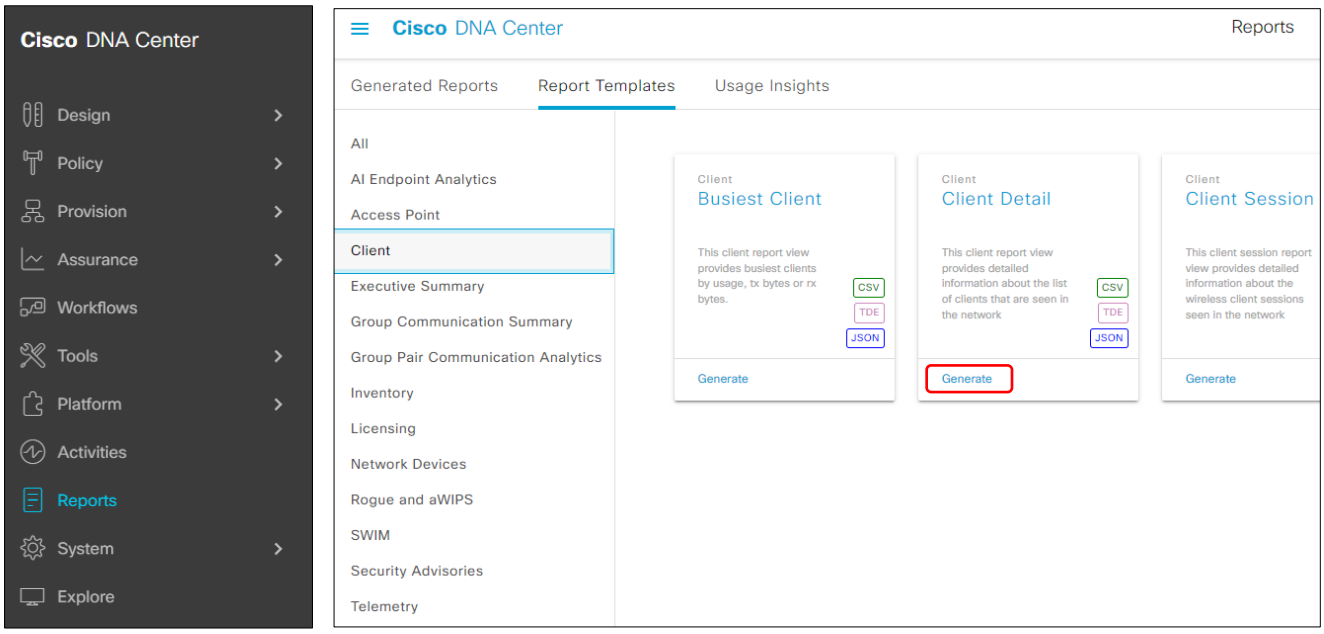
You can utilize data from the Reports feature in the Cisco DNA Center platform to derive insights into your network and its operation. By reporting this data in several formats and providing flexible scheduling and configuration options, both data and reports are also easily customized to meet your operational needs.

Creating a report on DNA Center's GUI is quite intuitive, however that might not be the case when it comes to report creation using APIs, thus, I have made this mini documentation to showcase the way how to create these reports using dnacentersdk package, and where to get all the needed parameters.

Note: this section explains in detail how to generate a **Client Details** report, however the overall idea applies to all other report types or templates.

3.2 Report Creation on DNA Center GUI

To create reports on DNA Center simply navigate to Reports on DNA Center

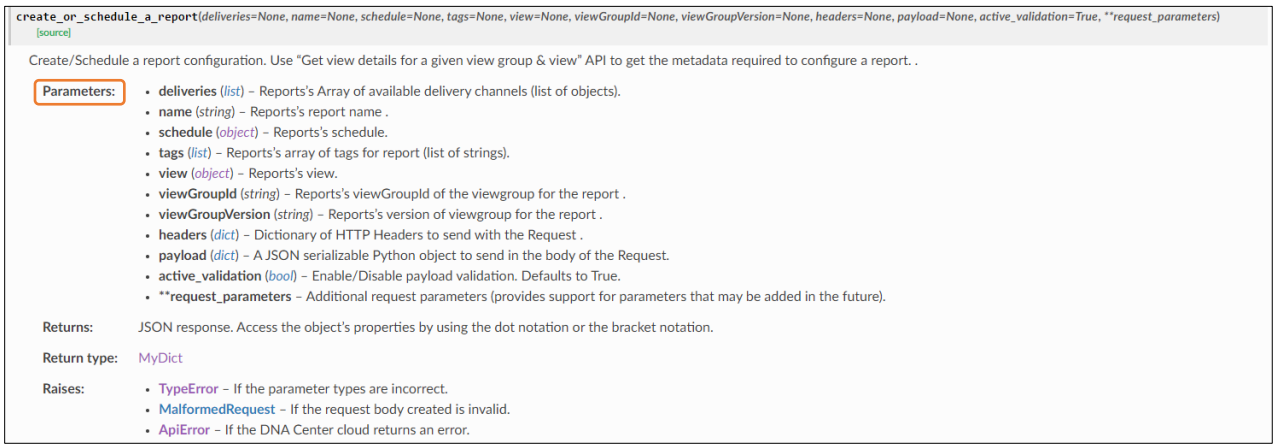


Choose a template category from the list on the left such as "Client", then generate a report from the available templates on the right side. The process is quite simple, a few parameters are required depending on the template such as: name of the report, location, SSID and others. Once you are done, the report will take a few seconds to generate, and could be found under "Generated Reports" section on the top left.

3.3 Report Creation Using dnacentersdk Report APIs

In this sub-section we will take a look at how to create Client Detail report using dnacentersdk package, lets first take a look at the API call on the documentation and explain every bit of it.

3.3.1 The API call and its parameters



`create_or_schedule_a_report`(*deliveries=None, name=None, schedule=None, tags=None, view=None, viewGroupId=None, viewGroupVersion=None, headers=None, payload=None, active_validation=True, **request_parameters*)
[source]

Create/Schedule a report configuration. Use "Get view details for a given view group & view" API to get the metadata required to configure a report. .

Parameters:
- **deliveries** (*list*) – Reports's Array of available delivery channels (list of objects).
- **name** (*string*) – Reports's report name .
- **schedule** (*object*) – Reports's schedule.
- **tags** (*list*) – Reports's array of tags for report (list of strings).
- **view** (*object*) – Reports's view.
- **viewGroupId** (*string*) – Reports's viewGroupId of the viewgroup for the report .
- **viewGroupVersion** (*string*) – Reports's version of viewgroup for the report .
- **headers** (*dict*) – Dictionary of HTTP Headers to send with the Request .
- **payload** (*dict*) – A JSON serializable Python object to send in the body of the Request.
- **active_validation** (*bool*) – Enable/Disable payload validation. Defaults to True.
- ****request_parameters** – Additional request parameters (provides support for parameters that may be added in the future).

Returns: JSON response. Access the object's properties by using the dot notation or the bracket notation.

Return type: MyDict

Raises:
- TypeError – If the parameter types are incorrect.
- MalformedRequest – If the request body created is invalid.
- ApiError – If the DNA Center cloud returns an error.

As we can see, this particular API call requires a number of parameters. The parameter can be found in the responses of the other API calls inside the red box

The required parameters are as follows:

- deliveries
- name
- schedule
- tags
- view
- viewGroupId
- viewGroupVersion
- Headers
- payload
- active_validation

All of these parameters values must follow a certain template and type, like *List, Dictionary, String, Object etc.*

Lets go over each one of them:

deliveries:

This section is for email and webhook deliveries, we could specify an email address for the report to be sent to or to associate it with a pre-configured webhook subscription profile.

```
deliveries = [

      {
         "type":"NOTIFICATION",
         "subscriptionId":"None",
         "notificationEndpoints":
         [
            {
               "type":"EMAIL",
               "emailAddresses":
               [
                  "jalluwat@cisco.com"
               ],
               "emailSubscriptionEndpointId":"None",
               "emailSubscriptionEndpointInstanceId":"None"
            }
         ],
         #"emailAttach":False #EMAIL ATTACHMENT IS FALSE BY DEFAULT, AND ONLY WORKS FOR PDF
REPORTS
      }
]
```

name:

Here we give the report a name. The name must be unique.

```

name = "Client Details - MyFirstReport"

```

schedule:

When would you like to schedule the report, now, later (one time) or recurring? With this parameter you could achieve that. In this example it's set to "Schedule Now".

```

schedule = {"type":"SCHEDULE_NOW"}

```

tags:

An array of tags for the report. This parameter isn't necessary to fill, and can be kept empty.

```
tags = []
```

view:

Please note: the shown variables and parameters in this example apply to Client Detail report template only

Under this parameter we add 3 main things which are: fields, filters, and format.

1. Fields: what fields are we including in the report, for example, a Client Details report could have up to 39 fields including things like: username, MAC address, IP address, RSSI and other client fields.
2. Filters: this section includes 4 main parameters:
    a. Time range (by default 3 hours) which is range of time the data is taken from
    b. Device type (wired, wireless or all)
    c. Band (5 GHz, 4 GHz, or both)
    d. Location (per site/s, building/s, both or global)
3. Format: here we specify the format of the report, for the Client Details report there 3 available formats to choose from: JSON, CSV, or Tableau Data Extract.
4. ViewID: each report template has a unique view ID, the one in the report is of Client Details report. I'll explain later where to get these data from.

Below the is the full ***view*** section

```
view = {
        "name":"Client Detail", #NAME OF THE REPORT
        "description":"Client Details via APIs",
        "fieldGroups":[
            {
            "fieldGroupName":"client_details",
            "fieldGroupDisplayName":"Client Details",
            "fields":[
               {
                  "name":"hostName",
                  "displayName":"Host Name"
               },
               {
                  "name":"macAddress",
                  "displayName":"MAC Address"
               },
               {
                  "name":"ipv4",
                  "displayName":"IPv4 Address"
               },
               {
                  "name":"averageHealthScore_median",
                  "displayName":"Median Health Score"
               },
               {
                  "name":"ssid",
                  "displayName":"SSID"
               }


```

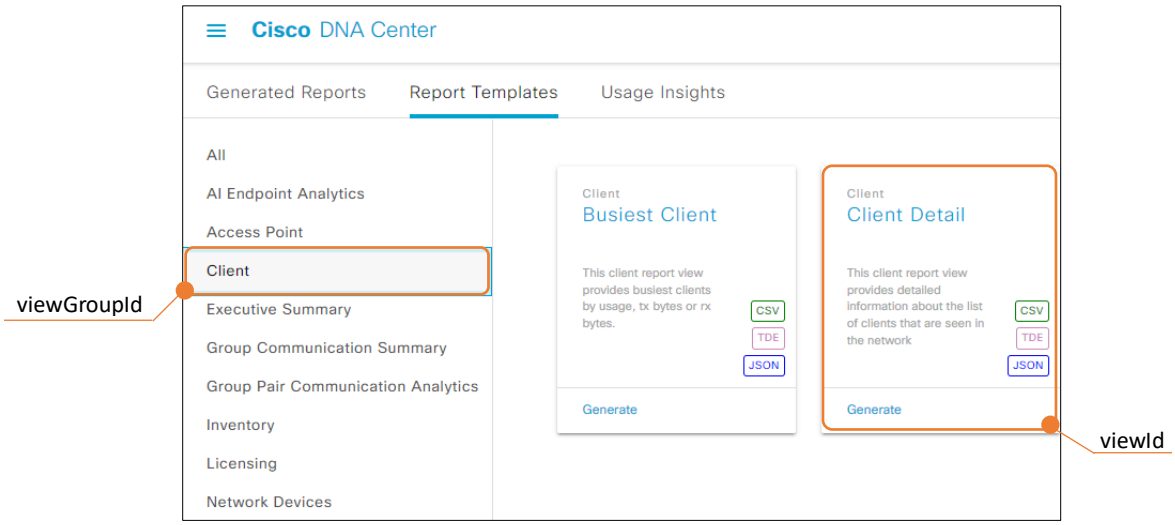
```
            ]
          }
        ],


"filters":[

            {
                "value": {"timeRangeOption":"LAST_3_HOURS"},#LAST 3 HOURS
                "displayName": "Time Range",
                "name": "TimeRange",
                "type": "TIME_RANGE"
            },

            {
                "type":"SINGLE_SELECT_ARRAY",
                "name":"DeviceType",
                "displayName":"Device Type",
                "value":[
                    {
                        "value":"Wireless",
                        "displayValue":"Wireless"
                    }
                ]
            },

            {
                "type":"MULTI_SELECT",
                "name":"Band",
                "displayName":"Band",
                "value":[
                    {
                        "value":"2.4",
                        "displayValue":"2.4"
                    },
                    {
                        "value":"5",
                        "displayValue":"5"
                    }
                ]
            },
            {
                "type":"MULTI_SELECT_TREE",
                "name":"Location",
                "displayName":"Location",
                "value":[
                    {
                        "value":"sample-global-site-id",
                        "displayValue":"Global"
                    }
                ]
            },
            {
                "type":"MULTI_SELECT",
                "name":"SSID",
                "displayName":"SSID",
                "value":[{

                        "value":"wsim-SSID",
                        "displayValue":"wsim-SSID" #WE COULD ADD HERE MULTIPLE SSID
                    }
```

```
                ]
            }
        ],
        "format":{
            "name":"JSON",
            "formatType":"JSON",
            #"default":true
        },
        "viewInfo":None,
        "viewId":"e8e66b17-4aeb-4857-af81-f472023bb05e" #THE UNIQUE VIEW ID FOR CLIENT
DETAIL REPORT
    }

```

view:

Each report template has its own ID called View Group ID, see the photo below that explains the idea behind these IDs.

```

viewGroupId = "d7afe5c9-4941-4251-8bf5-0fb643e90847" #THE UNIQUE ID FOR CLIENT CATEGORY

```



Note: each category has its own viewGroupId, and each template has its unique viewId

viewGroupVersion:

Report's version

```

viewGroupVersion = "2.0.0" #THIS IS THE LATEST VERSION BY THE TIME THIS DOC WAS MADE

```

headers:

Dictionary of HTTP Headers to send with the Request

```
1
2  headers = {} #THIS FIELD CAN BE LEFT BLANK
3
```

payload:

A JSON serializable Python object to send in the body of the Request

```
1
2  payload = {} #THIS FIELD CAN BE LEFT BLANK
3
```

active validation:

Enable/Disable payload validation. Defaults to True

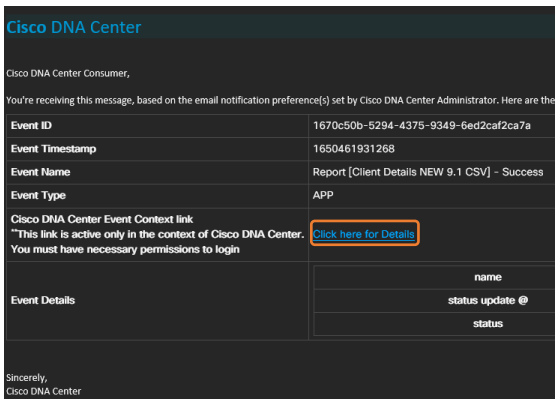
```
1
2  active_validation = False #I was running through errors when left it to its default value
3  “True”
```

A question that may raise now would be: where to get these parameters and script template from.

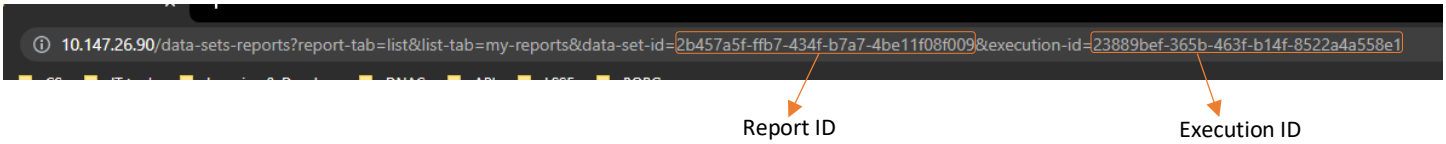
The answer is “from the other API calls”

My personal way of getting all the parameters I need are as follows:

1. Create a few reports **on DNA Center’s GUI** (I prefer creating different report formats i.e. JSON, CSV and others)
2. Have these reports sent to me via email
3. Click on the link found on the email to access the report



4. Copy the report ID number which is the first ID in the URL



5. Create a new code editor file, use the **get_a_scheduled_report()** API, and paste the report ID in the parenthesis as a parameter, as follows:

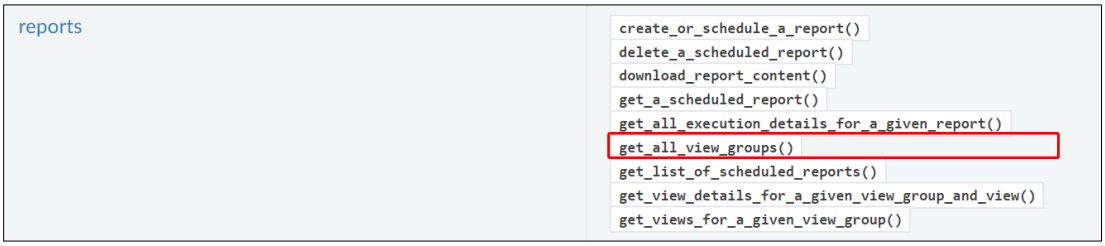
```
import urllib3
from dnacentersdk import DNACenterAPI
import json

urllib3.disable_warnings()

dnac = DNACenterAPI(username = '',
                    password = '',
                    base_url = 'https://{URL}:443',
                    version = '2.2.3.3',
                    verify = False)




data = dnac.reports.get_a_scheduled_report(report_id = "2b457a5f-ffb7-434f-b7a7-
4be11f08f009")

print(data)

```

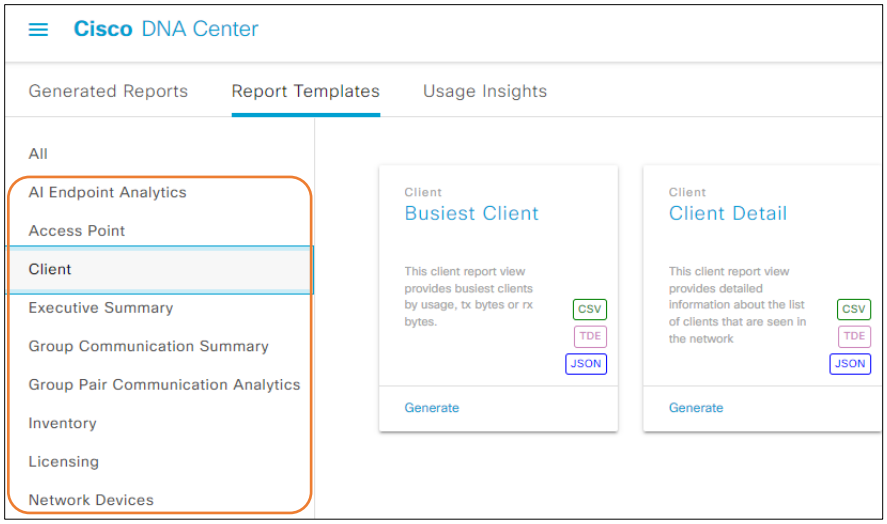
The output of this script is the full report template that you have generated previously on DNA Center. By having the template, you could do things like, adjusting the parameter, creating new reports, parsing through the output to get specific data, and much more.

You could use the **create_or_schedule_a_report()** to create another report, using the output template, and the information provided in previous steps

The mentioned steps are my personal way of getting a report template with all of its fields and variables, however, that not just it, there are many other report types that you could discover and adjust to your needs.

Try executing the following API call, which requires no parameters. The output of this call is basically the **viewGoupId**s of all of the report categories





Which gives you the possibility to explore other report types, not just client details one.

I think that's about wraps it up for this documentation, find below the link to my GitHub account to download the script, and another link to access the dnacentersdk documentation.

- GitHub DNA Center Report Creation Repo
- Dnacentersdk Documentation