Filtering and Generating JSON Data

Managing Data for Webex Teams, Network Devices and Services

**Author**

Yvan Rooseleer

* <[yvan.rooseleer@biasc.be](mailto:yvan.rooseleer@biasc.be)>, Belgian IT Academy Support Center
* <[yvan.rooseleer@odisee.be](mailto:yvan.rooseleer@odisee.be)>, Odisee University, Brussels

**Creation Date**

December 2020

**Session Dates IPD Week**

Part 1: DevNet Associate: Filtering JSON Data, 25 FEB 2021, 16:00   
Part 2: DevNet Associate: Generating JSON Data, 25 FEB 2021, 17:00

**Context**  
DevNet Assocate 1.0, Cisco Networking Academy

Practical Exercises: data formats (processing, filtering, generating), api calls, configuration management

**Topics**

Infrastructure as Code

YANG Data Modelling (basics)

Python programming

JSON, YAML XML

DNA Center (DNAC)

Docker (optional)

Ansible (optional)

Jenkins (optional)

**Tasks**

Reading, filtering and interpreting data in JSON format, coming from DNA Center, Ansible, Docker and Webex Teams.

Reading and processing data from a 2D Excel tabel

Transforming data into a JSON tree structure

Transformng JSON data to YAML

Transformng JSON data to XML

Using generated JSON data in API calls: e.g. creating spaces and members in Webex Teams, creating restconf and Ansible scritps.

**Part 1: DevNet Associate: Filtering JSON Data**

Description: This part covers some aspects of the DevNet Associate course. Join to learn how to interpret JSON data coming from Webex Teams API, DNAC API, Docker inspect, Ansible.

**Part 2: DevNet Associate: Generating JSON Data**

Description: This part covers some aspects of the DevNet Associate course. Join to learn how to generate JSON data for Webex Teams API, IP Devices and Network Services from a source in the Excel format.

**Level**

Intermediate

**Prerequisites**

Python programming basics

Understanding of  
-- Data modelling: YAML

-- Data formats: JSON, YAML, XML

-- Secure Shell

-- HTTPS/ SSL/ TLS

-- Webex Teams APIs

-- Restconf APIs

-- Ansible Playbooks

-- Docker

-- Jenkins CI/CD

## Part 1: DevNet Associate: Filtering JSON Data

### Preparing for Webex Teams API

**Target Structure**

The goal is experiment with basic data in a json tree structure format.

|  |
| --- |
| **### JSON FORMAT -- Keys with id will not be used in the processing example**  groups\_struc = {  "groups": [  { "group": { "group\_id": "G-A" , "group\_name": "DEVASC\_A" ,  "members": [  {"person\_id": "P-1" , "person\_name": "Noel", "email": "noel@odisee.be"},  {"person\_id": "P-2" , "person\_name": "Mary", "email": "mary@odisee.be"},  {"person\_id": "P-3" , "person\_name": "Jens", "email": "jens@odisee.be"}  ]  }  },  { "group": { "group\_id": "G-B" , "group\_name": "DEVASC\_B" ,  "members": [  {"person\_id": "P-4" ,"person\_name": "Ives", "email": "ives@odisee.be"},  {"person\_id": "P-5" ,"person\_name": "John", "email": "john@odisee.be"},  {"person\_id": "P-6" ,"person\_name": "Alec", "email": "alec@odisee.be"}  ]  }  },  { "group": { "group\_id": "G-C" , "group\_name": "DEVASC\_C" ,  "members": [  {"person\_id": "P-7" ,"person\_name": "Matt", "email": "matt@odisee.be"},  {"person\_id": "P-8" ,"person\_name": "Paul", "email": "paul@odisee.be"},  {"person\_id": "P-9" ,"person\_name": "Elvi", "email": "elvi@odisee.be"}  ]  }  }  ]  }  **Basic rules underlying the JSON structure**  ### RULES  groups\_struc = {} ### YANG CONTAINER  groups\_struc['groups'] = [] ### [group\_dict]  **group\_dict** = {} ### YANG LEAF {"group": {group\_name": "G" , "members": member\_list} }  group\_list = [] ### YANG LIST [group\_dict]  **member\_dict** = {} ### YANG LEAF {"person\_name": "x", "email": "y", "group":"z" }  member\_list = [] ### YANG LIST [member\_dict]  #  YANG leaves are comparable to a Python dict structure; YANG lists are comparable to an array (or list) in Python. |

**Experimenting with the target JSON structure and test data**

Experiment with the code below to train filtering JSON data.

|  |
| --- |
| print('------1---------')  print(type(groups\_struc))  print(groups\_struc)  print('------1A--------')  # converteer dict naar json  js\_groups = json.dumps(groups\_struc)  print(type(js\_groups))  print(js\_groups)  #print(json.dumps(groups\_struc, indent=2))  print('------2---------')  for g in groups\_struc["groups"]:  print('------2A--------')  print(type(g))  print(g)  print('------2B--------')  print(g["group"]["group\_name"])  print('------2C--------')  for p in g["group"]["members"]:  print(p["person\_name"] + " => " + p["email"])    print('------3---------')  print(groups\_struc.keys())  print('------3A---------')  print(groups\_struc["groups"][0].keys())  print('------3B---------')  print(groups\_struc["groups"][0]["group"].keys())  print('------3C---------')  print(groups\_struc["groups"][0]["group"]["members"][0].keys()) |

|  |
| --- |
| **Transforming DICT into JSON**  import json  js\_groups = json.dumps(groups\_struc)  print(json.dumps(groups\_struc, indent=2))  **Transforming JSON into YAML** import yaml  yaml\_data = yaml.dump(groups\_struc)  print(yaml\_data)  **Transforming JSON into XML**  from dicttoxml import dicttoxml  xml\_data = dicttoxml(groups\_struc)  print(xml\_data) |

### Preparing for DNA Center with Network Devices

**Target Structure**

The goal is experiment with basic data in a json tree structure format.

|  |
| --- |
| **### JSON FORMAT -- Keys with id not used in the processing example**  rack\_struc = {  "rack": [  { "device": { "dev\_id": "D1" ,  "dev\_name": "R1" ,  "role": "router" ,  "interfaces": [  {"interface": "GigabitEhternet1" ,  "ipaddress": "10.0.1.1",  "subnet\_mask": "255.255.255.0"},  {"interface": "GigabitEhternet2" ,  "ipaddress": "10.0.3.1",  "subnet\_mask": "255.255.255.0"},  {"interface": "GigabitEhternet3" ,  "ipaddress": "10.0.4.1",  "subnet\_mask": "255.255.255.0"}  ]  }  },  { "device": { "dev\_id": "D2" , "dev\_name": "C1" , "role": "core" ,  "interfaces": [  {"interface": "VLAN1" ,  "ipaddress": "10.0.1.2" ,  "subnet\_mask": "255.255.255.0"},  {"interface": "VLAN2" ,  "ipaddress": "10.0.2.1" ,  "subnet\_mask": "255.255.255.0"},  {"interface": "VLAN20" ,  "ipaddress": "10.0.20.1",  "subnet\_mask": "255.255.255.0"}  ]  }  },  { "device": { "dev\_id": "D3" , "dev\_name": "AC" , "role": "access" ,  "interfaces": [  {"interface": "VLAN2" ,  "ipaddress": "10.0.2.2",  "subnet\_mask": "255.255.255.0"}  ]  }  }  ]  }  **Basic rules underlying the JSON structure**  ### RULES  inventory\_dict = {} #### YANG CONTAINER  inventory\_list = []  rack\_struc = {} #### YANG CONTAINER  rack\_struc["rack"] = []  **dev\_dict** = {}  dev\_list = [] #### YANG LIST [dev\_dict]  **interface\_dict** = {} #### YANG LEAF {"interface": "gi0/1", "ipaddress": "1.2.2.1", "subnetmask": "255.255.255.0"}  interface\_list = [] #### YANG LIST [interface\_dict]  #  *YANG leaves are comparable to a Python dict structure; YANG lists are comparable to an array (or list) in Python.* |

**Experimenting with the target JSON structure and test data**

|  |
| --- |
| **print('------1---------')**  print(type(rack\_struc))  print(rack\_struc)  #print('------1A--------')  js\_struc = json.dumps(rack\_struc)  #print(type(js\_struc))  #print(js\_struc)  #print(json.dumps(rack\_struc, indent=8))  **print('------1B--------')**  g = rack\_struc["rack"][0]  print(type(g))  print(g["device"].keys())  **print('------2---------')**  for g in rack\_struc["rack"]:  **print('------2A--------')**  print(type(g))  print(g)  print(g["device"]["dev\_name"])  for p in g["device"]["interfaces"]:  print(p["ipaddress"])    **print('------3---------')**  print("Keys device")  print(g["device"].keys())  **print('------3A---------')**  print("Keys interfaces")  print(g["device"]["interfaces"][0].keys()) |

**Experimenting with the target JSON structure and test data**

|  |
| --- |
| **Transforming DICT into JSON**  js\_struc = json.dumps(rack\_struc)  print(type(js\_struc))  #print(js\_struc)  print(json.dumps(rack\_struc, indent=4))  **Transforming JSON into YAML** import yaml  yaml\_data = yaml.dump(rack\_struc)  print(yaml\_data)  **Transforming JSON into XML**  from dicttoxml import dicttoxml  xml\_data = dicttoxml(rack\_struc)  print(xml\_data) |

### Network Service Examples

**Target Structure**

The goal is experiment with basic data in a json tree structure format.

|  |
| --- |
| **### JSON FORMAT -- Keys with id not used in the processing example**  rack\_struc = {  "rack": [  { "server": { "dev\_id": "S1" , "server\_name": "svr1" , "domain": "biasc.be", "ip-address": "10.2.3.1" ,  "os": "linux" , "server\_type": "vm" ,  "services": [  {"service": "ad" , "service\_type": "vm", "protocol": "tcp", "port": "389"},  {"service": "dns", "service\_type": "vm", "protocol": "udp", "port": "53"},  {"service": "ntp", "service\_type": "vm", "protocol": "tcp", "port": "123"}  ]  }  },  { "server": { "dev\_id": "S2" , "server\_name": "svr2" , "domain": "biasc.be", "ip-address": "10.2.3.2" ,  "os": "linux" , "server\_type": "vm" ,  "services": [  {"service": "flask", "service\_type": "vm", "protocol": "tcp", "port": "8089" },  {"service": "db" , "service\_type": "vm", "protocol": "tcp", "port": "1521" }  ]  }  },  { "server": { "dev\_id": "S3" , "server\_name": "svr3" , "domain": "biasc.be" , "ip-address": "10.2.3.3",  "os": "linux" , "server\_type": "vm" ,  "services": [  {"service": "dns" , "service\_type": "vm", "protocol": "tcp", "port": "8089" },  {"service": "ntp" , "service\_type": "vm", "protocol": "tcp", "port": "8089" },  {"service": "dhcp", "service\_type": "docker", "protocol": "udp", "port": "67" }  ]  }  }  ]  }  **Basic rules underlying the JSON structure**  ### RULES  inventory\_dict = {}  inventory\_list = [] #### [inventory\_dict]  **service\_dict** = {}  service\_list = [] #### [service\_dict]  **dev\_dict** = {}  dev\_list = [] #### [dev\_dict]  rack\_struc = {} #### {dev\_dict\_list}  rack\_struc["rack"] = []  #  *YANG leaves are comparable to a Python dict structure; YANG lists are comparable to an array (or list) in Python.* |

**Experimenting with the target JSON structure and test data**

|  |
| --- |
| **print('------1---------')**  **#print(devices\_struc)**  **#print('------1A--------')**  **js\_groups = json.dumps(rack\_struc)**  **print(js\_groups)**  **print(json.dumps(rack\_struc, indent=2))**  **print('------2---------')**  **for g in devices\_struc["rack"]:**  **print('------2A--------')**  **print(type(g))**  **print(g)**  **print(g["server"]["services"])**  **for p in g["server"]["services"]:**  **print(p)**    **print('------3---------')**  **print(devices\_struc.keys())**  **print('------3A---------')**  **print(devices\_struc["rack"][0].keys())**  **print('------3B---------')**  **print(devices\_struc["rack"][0]["server"].keys())**  **print('------3C---------')**  **print(devices\_struc["rack"][0]["server"]["services"][0].keys())** |

**Experimenting with the target JSON structure and test data**

|  |
| --- |
| **Transforming DICT into JSON**  js\_struc = json.dumps(devices\_struc)  print(type(js\_struc))  #print(js\_struc)  print(json.dumps(devices\_struc, indent=4))  **Transforming JSON into YAML** import yaml  yaml\_data = yaml.dump(devices\_struc)  print(yaml\_data)  **Transforming JSON into XML**  from dicttoxml import dicttoxml  xml\_data = dicttoxml(devices\_struc)  print(xml\_data) |

**Generating JSON data structures based on an Excel spreadsheet**

Here is the code for generating the JSON data structures based on the excel spreadsheet webex\_groups.xlsx.

|  |
| --- |
| ### FUNCTIONS TO CREATE (A) list of servers and (B) list of services  ### Using an excel file as input for a Python script  ### This script creates a JSON structure based on information an excel spreadsheet  ### You have to understand Python dict and Python lists  ### Contact yvan.rooseleer@biasc.be if you have questions  ### It's up to you to ### WRITE YOUR OWN CODE ### where this is indicated  ### Validate your JSON structure  ### Use print statements to understand return values and return types  ### There is a simplified solution (main2) and a more difficult solution (main)  ###  import xlrd  import json  ### RULES  inventory\_dict = {} #### {"server\_name": "srv1", "domain": "biasc.be", "ipaddress": "10.1.1.1", "os": "xe" , "server\_type": "file server"  inventory\_list = [] #### [inventory\_dict]  service\_dict = {} #### {"service": "ntp", "service\_type": "vm", "protocol": "udp", "port"= "123"}  service\_list = [] #### [service\_dict]  dev\_dict = {} #### {"server": {server\_name": "n", "domain": "r", "services": service\_list}}  dev\_list = [] #### [dev\_dict]  rack\_struc = {} #### {dev\_dict\_list}  rack\_struc["rack"] = []  ###  def find\_all\_services(xlf):  ### READ EXCEL FILE AND RETURN NUMBER OF ROWS  wb = xlrd.open\_workbook(xlf)  sheet = wb.sheet\_by\_index(0)  number\_rows = sheet.nrows  loc\_interfaces = []  for r in range(number\_rows):  if r > 0: ### first row contains columns names  COL\_A = sheet.cell\_value(r, 0) #### column A  COL\_B = sheet.cell\_value(r, 1) #### column B  COL\_C = sheet.cell\_value(r, 2) #### column C  COL\_D = sheet.cell\_value(r, 3) #### column D  COL\_E = sheet.cell\_value(r, 4) #### column E  COL\_F = sheet.cell\_value(r, 5) #### column F  COL\_G = sheet.cell\_value(r, 6) #### column G  COL\_H = sheet.cell\_value(r, 7) #### column H  COL\_I = sheet.cell\_value(r, 8) #### column I  inventory\_dict["server"] = COL\_A  inventory\_dict["domain"] = COL\_B  inventory\_dict["ipaddress"] = COL\_C  inventory\_dict["os"] = COL\_D  inventory\_dict["server\_type"] = COL\_E  inventory\_dict["service"] = COL\_F  inventory\_dict["service\_type"] = COL\_G  inventory\_dict["protocol"] = COL\_H  inventory\_dict["port"] = COL\_I  loc\_interfaces.append(inventory\_dict.copy()) # need to use copy()  return loc\_interfaces  def make\_list\_of\_servers(loc\_list):  loc\_g = []  mem = {}  g = {}  for rec in loc\_list:  g["server\_name"] = rec["server"]  g["domain"] = rec["domain"]  g["ipaddress"] = rec["ipaddress"]  g["os"] = rec["os"]  g["service\_type"] = rec["service\_type"]  if mem != g["server\_name"]:  loc\_g.append(g.copy()) # need to use copy()  mem = g["server\_name"]  #for rec in loc\_g:  #print(rec)  #del loc\_g[0] ### if last item copied as first item  return loc\_g  def attach\_services(loc\_server, loc\_inventory):  loc\_srv\_dict = {}  loc\_list = [loc\_srv\_dict]  for item in loc\_inventory:  if item["server"] == loc\_server:  if item["server"] != None:  loc\_srv\_dict["service"] = item["service"]  loc\_srv\_dict["service\_type"] = item["service\_type"]  loc\_srv\_dict["protocol"] = item["protocol"]  loc\_srv\_dict["port"] = item["port"]  loc\_list.append(loc\_srv\_dict.copy()) # need to use copy()  del loc\_list[0] ### if last item copied as first item  return loc\_list    #### MAIN ####  def main(): ### role added  inventory\_list = find\_all\_services("servers.xlsx")  dev\_list = make\_list\_of\_servers(inventory\_list)  for device\_rec in dev\_list:  srvc\_list = attach\_services(device\_rec["server\_name"], inventory\_list)  dev\_dict["server"] = { "server": device\_rec , "services": srvc\_list }  rack\_struc["rack"].append(dev\_dict["server"])  js\_rack = json.dumps(rack\_struc)  #### execute main() when called directly  if \_\_name\_\_ == '\_\_main\_\_':  main() |

### More Examples from Webex Teams API

To be continued

### Examples from Ansible

To be continued

### Examples from Docker Inspect

To be continued

## 

## Part 2: DevNet Associate: Generating JSON Data

There are three tasks:

* Webex Teams
* Network Devices
* Network Services

The tasks have an increasing complexity, you should start with task 1 (Webex teams Groups)

**Task 1 -- Webex Teams**  
Reading and transforming an Excel sheet with groups and users into a JSON tree structure

Creating, validating and testing a JSON structure

Selecting and filtering JSON data

Transforming JSON data into YAMl and XML

Using Webex Teams API calls to create groups (spaces) and members (users)

**Task 2 -- Network Devices**

Reading and transforming an Excel sheet with new network devices and IP addresses into a JSON tree structure

Creating, validating and testing a JSON structure

Selecting and filtering JSON data

Transforming JSON data into YAMl and XML

Creating restconf script to configure network devices

Creating ansible script to configure network devices (optional)

**Task 3 -- Network Services**

Reading and transforming an Excel sheet with a list of new servers and services into a JSON tree structure

Creating, validating and testing a JSON structure

Creating ansible playbook to configure services on servers (optional)

Creating dockerfile to generate specific images for services (optional)

Using jenkins to pipeline deployment of specific services (optional)

# 

# TASK 1 - WEBEX TEAMS

Reading and transforming an Excel sheet with groups and users into a JSON tree structure

Creating, validating and testing a JSON structure

Selecting and filtering JSON data

Transforming JSON data into YAMl and XML

Using Webex Teams API calls to create groups (spaces) and members (users)

**Source File**

Here is an example of a source file with three columns (groups, name, email)

|  |  |  |
| --- | --- | --- |
| **group** | **name** | **email** |
| GROUP\_ALPHA | Vincent Cassata | vincent.cassata@student.bxl.be |
| GROUP\_ALPHA | Giovanni Di Tulio | Giovanni .ditullio@student.bxl.be |
| GROUP\_ALPHA | Milan Vandevelde | milan.vandevelde@student.bxl.be |
| GROUP\_ALPHA | Tomas Vertessen | tomas.vertessen@student.bxl.be |
| GROUP\_ALPHA | Mehdi Dahli | mehdi.dahli@student.bxl.be |
| GROUP\_KAPPA | Ur Salangpour | ur.salangpour@student.bxl.be |
| GROUP\_KAPPA | Mon Gallin | mon.gallin@student.bxl.be |
| GROUP\_KAPPA | Artur Ikiya | artur.lkiya@student.bxl.be |
| GROUP\_KAPPA | Bram Vanbever | bram.vanbever@student.bxl.be |
| GROUP\_KAPPA | JR Ibara | jr.ibara@student.bxl.be |
| GROUP\_DELTA | Jona Ferbiest | jona.ferbiest@student.bxl.be |
| GROUP\_DELTA | Bart Siperius | bart.siperius@student.bxl.be |
| GROUP\_DELTA | Joren Huysegoms | joren.huysegoms2@student.bxl.be |
| GROUP\_DELTA | Sam Bulduk | sam.bulduk@student.bxl.be |
| GROUP\_DELTA | Ferre Van Malder | ferre.vanmalder@student.bxl.be |
| GROUP\_DELTA | Mikail Defossez | mikail.defossez@student.bxl.be |

**File name: webex\_groups.xlsx**

**Target Structure**

The goal is to read the excel data and generate a json tree structure format according to the example below:

|  |
| --- |
| **### JSON FORMAT -- Keys with id will not be used in the processing example**  groups\_struc = {  "groups": [  { "group": { "group\_id": "G-A" , "group\_name": "DEVASC\_A" ,  "members": [  {"person\_id": "P-1" , "person\_name": "Noel", "email": "noel@odisee.be"},  {"person\_id": "P-2" , "person\_name": "Mary", "email": "mary@odisee.be"},  {"person\_id": "P-3" , "person\_name": "Jens", "email": "jens@odisee.be"}  ]  }  },  { "group": { "group\_id": "G-B" , "group\_name": "DEVASC\_B" ,  "members": [  {"person\_id": "P-4" ,"person\_name": "Ives", "email": "ives@odisee.be"},  {"person\_id": "P-5" ,"person\_name": "John", "email": "john@odisee.be"},  {"person\_id": "P-6" ,"person\_name": "Alec", "email": "alec@odisee.be"}  ]  }  },  { "group": { "group\_id": "G-C" , "group\_name": "DEVASC\_C" ,  "members": [  {"person\_id": "P-7" ,"person\_name": "Matt", "email": "matt@odisee.be"},  {"person\_id": "P-8" ,"person\_name": "Paul", "email": "paul@odisee.be"},  {"person\_id": "P-9" ,"person\_name": "Elvi", "email": "elvi@odisee.be"}  ]  }  }  ]  }  **Basic rules underlying the JSON structure**  ### RULES  groups\_struc = {} ### YANG CONTAINER  groups\_struc['groups'] = [] ### [group\_dict]  **group\_dict** = {} ### YANG LEAF {"group": {group\_name": "G" , "members": member\_list} }  group\_list = [] ### YANG LIST [group\_dict]  **member\_dict** = {} ### YANG LEAF {"person\_name": "x", "email": "y", "group":"z" }  member\_list = [] ### YANG LIST [member\_dict]  #  YANG leaves are comparable to a Python dict structure; YANG lists are comparable to an array (or list) in Python. |

**Generating JSON data structures based on an Excel spreadsheet**

Here is the code for generating the JSON data structures based on the excel spreadsheet webex\_groups.xlsx.

|  |
| --- |
| ### FUNCTIONS TO CREATE (A) GROUPS AND (B) GROUP MEMBERSHIPS  ### Using an excel file as input for a Python script  ###  import xlrd  import json  ### RULES  member\_dict = { "person\_name": "x", "email": "y", "group":"z" }  member\_list = [] ### [member\_dict]  group\_dict = {} ### {"group": {group\_name": "G" , "members": member\_list} }  group\_list = [] ### [group\_dict]  groups\_struc = {}  groups\_struc['groups'] = [] ### [group\_dict]  ###  def find\_all\_persons\_and\_groups(xlf):  ### READ EXCEL FILE AND RETURN NUMBER OF ROWS  wb = xlrd.open\_workbook(xlf)  sheet = wb.sheet\_by\_index(0)  number\_rows = sheet.nrows  loc\_member\_list = []  for r in range(number\_rows):  if r > 0: ### first row contains columns names  COL\_A = sheet.cell\_value(r, 0) #### column A  COL\_B = sheet.cell\_value(r, 1) #### column B  COL\_C = sheet.cell\_value(r, 2) #### column C  member\_dict["group"] = COL\_A  member\_dict["person\_name"] = COL\_B  member\_dict["email"] = COL\_C  loc\_member\_list.append(member\_dict.copy())  return loc\_member\_list  ###  def make\_list\_of\_groups(loc\_dict):  loc\_g = []  mem = None  for rec in loc\_dict:  g = rec["group"]  if mem != g:  loc\_g.append(g)  mem = g  return loc\_g  ###  def attach\_members\_to\_groups(loc\_group, loc\_members):  loc\_m\_dict = {}  loc\_m\_list = [loc\_m\_dict]  for membr in loc\_members:  if membr["group"] == loc\_group:  if membr["person\_name"] != None:  loc\_m\_dict["person\_name"] = membr["person\_name"]  loc\_m\_dict["email"] = membr["email"]  loc\_m\_list.append(loc\_m\_dict.copy())  return loc\_m\_list    #### MAIN ####  def main():  #member\_list = find\_all\_persons\_and\_groups("webex\_groups\_gr.xlsx")  #member\_list = find\_all\_persons\_and\_groups("webex\_devops.xlsx")  member\_list = find\_all\_persons\_and\_groups("webex\_groups.xlsx")  group\_list = make\_list\_of\_groups(member\_list)  all\_members = []  for group\_rec in group\_list:  all\_members = attach\_members\_to\_groups(group\_rec, member\_list)  del all\_members[0] #### delete the first element, which is a copy of the last element  group\_dict["group"] = { "group": {"group\_name": group\_rec , "members": all\_members }}  groups\_struc['groups'].append(group\_dict["group"])  js\_groups = json.dumps(groups\_struc)    #### execute main() when called directly  if \_\_name\_\_ == '\_\_main\_\_':  main() |

**Webex Teams API**

Here is a code snippet showing how you can use the JSON data to create spaces and groups in Webex Teams. While you are running the code, check the result in Webex teams.

|  |
| --- |
| ### Access Token 12 hours: **https://developer.webex.com/docs/api/getting-started** ### (login required)  current\_access\_token = "Njk3YWM0YmMtYTllOC00ZWRlLWIxNDgtZDc4ODcxYzdjMDU4ZWQ0OGE3ZmYtNzZk\_PF84\_e4d4112d-2548-4a47-810e-04fe45ea181f" ### add your own bearer token  ### ADD NEW SPACES AND MEMBERS TO WEBEX TEAMS  **### THIS CODE ONLY WORKS IF YOU ARE ABLE TO GENERATE A CORRECT GROUPS\_STRUC**  import requests  import json  ###  from **webexteamssdk** import WebexTeamsAPI  api = WebexTeamsAPI(access\_token=current\_access\_token)  ###  print("Creating spaces + members --> from Excel spreadsheet in the previous cell")  access\_token = current\_access\_token  **def main2(): # using rest api**  url = 'https://api.ciscospark.com/v1/rooms'  headers = { 'Authorization': 'Bearer {}'.format(access\_token),  'Content-Type': 'application/json' }  for rec in groups\_struc["groups"]:  create\_group\_name = rec["group"]["group\_name"]  payload\_space={"title": create\_group\_name}  if payload\_space["title"] != None: ### avoid errors if room title unknown  res\_space = requests.post(url, headers=headers, json=payload\_space)  #print(payload\_space)  #print(res\_space.text)  if res\_space.status\_code < 300:  NEW\_SPACE\_ID = res\_space.json()["id"]  #print(type(NEW\_SPACE\_ID))  #print(NEW\_SPACE\_ID)  for mbr in rec["group"]["members"]:  room\_id = NEW\_SPACE\_ID  person\_email = mbr["email"]  url2 = 'https://api.ciscospark.com/v1/memberships'  payload\_member = {'roomId': room\_id, 'personEmail': person\_email}  #print(payload\_member)  res\_member = requests.post(url2, headers=headers, json=payload\_member)  **def main(): # using webxteamssdk**  for rec in groups\_struc["groups"]:  # Create a new demo room  demo\_room = api.rooms.create(rec["group"]["group\_name"])  # Add people to the new demo room  for email in rec["group"]["members"]:  api.memberships.create(demo\_room.id, personEmail=email["email"])  **#### execute main() when called directly**  **if \_\_name\_\_ == "\_\_main\_\_":**  **main2() ### or main()** |

# TASK 2 - IP DEVICES

Reading and transforming an Excel sheet with new network devices and IP addresses into a JSON tree structure

Creating, validating and testing a JSON structure

Selecting and filtering JSON data

Transforming JSON data into YAMl and XML

Creating restconf script to configure network devices

Creating ansible script to configure network devices (optional)

**Source File**

Here is an example of a source file with five columns (device, role, interface, ip address, subnet mask).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **device** | **role** | **interface** | **ipaddress** | **subnetmask** |
| RTR1 | router | GigabitEthernet 0 | 192.0.2.254 | 255.255.255.0 |
| RTR1 | router | GigabitEthernet 1 | 10.0.1.1 | 255.255.255.0 |
| RTR1 | router | GigabitEthernet 2 | 10.0.2.1 | 255.255.255.0 |
| MLS1 | core switch | VLAN 1 | 10.0.1.2 | 255.255.255.0 |
| MLS1 | core switch | VLAN 2 | 10.0.2.1 | 255.255.255.0 |
| MLS2 | core switch | VLAN 1 | 10.0.1.3 | 255.255.255.0 |
| MLS2 | core switch | VLAN 2 | 10.0.2.2 | 255.255.255.0 |
| ASW2 | access switch | VLAN 1 | 10.0.1.2 | 255.255.255.0 |
| ASW3 | access switch | VLAN 1 | 10.0.1.3 | 255.255.255.0 |
| ASW4 | access switch | VLAN 1 | 10.0.1.4 | 255.255.255.0 |
| ASW5 | access switch | VLAN 1 | 10.0.1.5 | 255.255.255.0 |
| ASW6 | access switch | VLAN 2 | 10.0.2.6 | 255.255.255.0 |
| ASW7 | access switch | VLAN 2 | 10.0.2.7 | 255.255.255.0 |
| ASW8 | access switch | VLAN 2 | 10.0.2.8 | 255.255.255.0 |
| ASW9 | access switch | VLAN 2 | 10.0.2.9 | 255.255.255.0 |

**File name: ipdevices.xlsx**

**Target Structure**

The goal is to read the excel data and generate a json tree structure format according to the example below:

|  |
| --- |
| **### JSON FORMAT -- Keys with id not used in the processing example**  rack\_struc = {  "rack": [  { "device": { "dev\_id": "D1" ,  "dev\_name": "R1" ,  "role": "router" ,  "interfaces": [  {"interface": "GigabitEhternet1" ,  "ipaddress": "10.0.1.1",  "subnet\_mask": "255.255.255.0"},  {"interface": "GigabitEhternet2" ,  "ipaddress": "10.0.3.1",  "subnet\_mask": "255.255.255.0"},  {"interface": "GigabitEhternet3" ,  "ipaddress": "10.0.4.1",  "subnet\_mask": "255.255.255.0"}  ]  }  },  { "device": { "dev\_id": "D2" , "dev\_name": "C1" , "role": "core" ,  "interfaces": [  {"interface": "VLAN1" ,  "ipaddress": "10.0.1.2" ,  "subnet\_mask": "255.255.255.0"},  {"interface": "VLAN2" ,  "ipaddress": "10.0.2.1" ,  "subnet\_mask": "255.255.255.0"},  {"interface": "VLAN20" ,  "ipaddress": "10.0.20.1",  "subnet\_mask": "255.255.255.0"}  ]  }  },  { "device": { "dev\_id": "D3" , "dev\_name": "AC" , "role": "access" ,  "interfaces": [  {"interface": "VLAN2" ,  "ipaddress": "10.0.2.2",  "subnet\_mask": "255.255.255.0"}  ]  }  }  ]  }  **Basic rules underlying the JSON structure**  ### RULES  inventory\_dict = {} #### YANG CONTAINER  inventory\_list = []  rack\_struc = {} #### YANG CONTAINER  rack\_struc["rack"] = []  **dev\_dict** = {}  dev\_list = [] #### YANG LIST [dev\_dict]  **interface\_dict** = {} #### YANG LEAF {"interface": "gi0/1", "ipaddress": "1.2.2.1", "subnetmask": "255.255.255.0"}  interface\_list = [] #### YANG LIST [interface\_dict]  #  *YANG leaves are comparable to a Python dict structure; YANG lists are comparable to an array (or list) in Python.* |

**Generating JSON data structures based on an Excel spreadsheet**

Here is the code for generating the JSON data structures based on the excel spreadsheet webex\_groups.xlsx.

|  |
| --- |
| ### FUNCTIONS TO CREATE (A) list of devices and (B) list of device interfaces  ### Using an excel file as input for a Python script  ### This script creates a JSON structure based on information an excel spreadsheet  ### You have to understand Python dict and Python lists  ### Contact yvan.rooseleer@biasc.be if you have questions  ### It's up to you to ### WRITE YOUR OWN CODE ### where this is indicated  ### Validate your JSON structure  ### Use print statements to understand return values and return types  ### There is a simplified solution (main2) and a more difficult solution (main)  ###  import xlrd  import json  ###  ### RULES  inventory\_dict = {} #### {"interface": "gi0/1", "ip\_address": "1.2.2.1", "subnet\_mask": "255.255.255.0", "device": "dv"  inventory\_list = [] #### [inventory\_dict]  interface\_dict = {} #### {"interface": "gi0/1", "ip\_address": "1.2.2.1", "subnet\_mask": "255.255.255.0"}  interface\_list = [] #### [interface\_dict]  dev\_dict = {} #### {"device": {dev\_name": "n", "role": "r", interfaces": interface\_list}}  dev\_list = [] #### [dev\_dict]  rack\_struc = {} #### {dev\_dict\_list}  rack\_struc["rack"] = []  ####  def find\_all\_device\_interfaces(xlf):  ### READ EXCEL FILE AND RETURN NUMBER OF ROWS  wb = xlrd.open\_workbook(xlf)  sheet = wb.sheet\_by\_index(0)  number\_rows = sheet.nrows  loc\_interfaces = []  for r in range(number\_rows):  if r > 0: ### first row contains columns names  COL\_A = sheet.cell\_value(r, 0) #### column A  COL\_B = sheet.cell\_value(r, 1) #### column B  COL\_C = sheet.cell\_value(r, 2) #### column C  COL\_D = sheet.cell\_value(r, 3) #### column D  COL\_E = sheet.cell\_value(r, 4) #### column E  inventory\_dict["device"] = COL\_A  inventory\_dict["role"] = COL\_B  inventory\_dict["interface"] = COL\_C  inventory\_dict["ipaddress"] = COL\_D  inventory\_dict["subnetmask"] = COL\_E  loc\_interfaces.append(inventory\_dict.copy()) # need to use copy()  return loc\_interfaces  def make\_list\_of\_devices\_and\_roles(loc\_list):  loc\_g = []  mem = {}  g = {}  for rec in loc\_list:  g["dev\_name"] = rec["device"]  g["role"] = rec["role"]  if mem != g["dev\_name"]:  loc\_g.append(g.copy()) # need to use copy()  mem = g["dev\_name"]  #for rec in loc\_g:  #print(rec)  #del loc\_g[0] ### if last item copied as first item  return loc\_g  def make\_list\_of\_devices(loc\_list):  loc\_g = []  mem = None  for rec in loc\_list:  g = rec["device"]  if mem != g:  loc\_g.append(g)  mem = g  return loc\_g  def attach\_interfaces\_to\_devices(loc\_device, loc\_inventory):  loc\_dev\_dict = {}  loc\_list = [loc\_dev\_dict]  for item in loc\_inventory:  if item["device"] == loc\_device:  if item["device"] != None:  loc\_dev\_dict["interface"] = item["interface"]  loc\_dev\_dict["ipaddress"] = item["ipaddress"]  loc\_dev\_dict["subnetmask"] = item["subnetmask"]  loc\_list.append(loc\_dev\_dict.copy()) # need to use copy()  del loc\_list[0] ### if last item copied as first item  return loc\_list    #### MAIN ####  def main2(): ### no role for devices, only device name  inventory\_list = find\_all\_device\_interfaces("devices\_ip.xlsx")  dev\_list = make\_list\_of\_devices(inventory\_list)  for device\_rec in dev\_list:  intf\_list = attach\_interfaces\_to\_devices(device\_rec, inventory\_list)  dev\_dict["device"] = { "device": {"dev\_name": device\_rec , "interfaces": intf\_list }}  rack\_struc["rack"].append(dev\_dict["device"])  js\_rack = json.dumps(rack\_struc)  def main(): ### role added  inventory\_list = find\_all\_device\_interfaces("devices\_ip.xlsx")  dev\_list = make\_list\_of\_devices\_and\_roles(inventory\_list)  for device\_rec in dev\_list:  intf\_list = attach\_interfaces\_to\_devices(device\_rec["dev\_name"], inventory\_list)  dev\_dict["device"] = { "device": device\_rec , "interfaces": intf\_list }  rack\_struc["rack"].append(dev\_dict["device"])  js\_rack = json.dumps(rack\_struc)  #### execute main() when called directly  if \_\_name\_\_ == '\_\_main\_\_':  main() ### alternative: main2() |

# TASK 3 - Network Services

Reading and transforming an Excel sheet with a list of new servers and services into a JSON tree structure

Creating, validating and testing a JSON structure

Creating ansible playbook to configure services on servers (optional)

Creating dockerfile to generate specific images for services (optional)

Using jenkins to pipeline deployment of specific services (optional)

**Source File**

Here is an example of a source file with nine columns (server, domain,ip address, os, server type, service, protocol, port).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **server** | **domain** | **ipadress** | **OS** | **server\_type** | **service** | **service\_type** | **protocol** | **port** |
| srv1 | [biasc.be](http://biasc.be) | 192.0.2.254 | windows | directory server | ldap/ad | vm | tcp | 389 |
| srv1 | [biasc.be](http://biasc.be) | 10.0.1.1 | linux | name server | dns | vm | udp | 53 |
| srv1 | [biasc.be](http://biasc.be) | 10.0.2.1 | linux | mail server | smtp | vm | tcp | 25 |
| srv2 | [biasc.be](http://biasc.be) | 10.0.1.2 | linux | time server | ntp | docker | udp | 123 |
| srv2 | [biasc.be](http://biasc.be) | 10.0.2.2 | linux | web server | flask | docker | tcp | 8089 |
| srv2 | [biasc.be](http://biasc.be) | 10.0.2.3 | linux | configuration management | ansible | docker | ssh | 22 |
| srv2 | [biasc.be](http://biasc.be) | 10.0.2.4 | linux | CI/CD | jenkins | docker | tcp | 8080 |
| srv3 | [biasc.be](http://biasc.be) | 10.0.1.3 | linux | name server | dns | docker | udp | 53 |
| srv4 | [biasc.be](http://biasc.be) | 10.0.1.4 | linux | file server | ftp | vm | tcp | 20/21 |
| srv5 | [biasc.be](http://biasc.be) | 10.0.1.5 | linux | application server | oracle app | vm | tcp | 8081 |
| srv6 | [biasc.be](http://biasc.be) | 10.0.1.6 | linux | application server | oracle app | vm | tcp | 8081 |
| srv7 | [biasc.be](http://biasc.be) | 10.0.1.7 | linux | database server | oracle db | vm | tcp | 1521 |
| srv8 | [biasc.be](http://biasc.be) | 10.0.1.8 | linux | database server | oracle db | vm | tcp | 1521 |
| srv9 | [biasc.be](http://biasc.be) | 10.0.1.9 | linux | addressing server | dhcp | docker | udp | 67 |

**Target Structure**

The goal is to read the excel data and generate a json tree structure format according to the example below:

|  |
| --- |
| **### JSON FORMAT -- Keys with id not used in the processing example**  rack\_struc = {  "rack": [  { "server": { "dev\_id": "S1" , "server\_name": "svr1" , "domain": "biasc.be", "ip-address": "10.2.3.1" ,  "os": "linux" , "server\_type": "vm" ,  "services": [  {"service": "ad" , "service\_type": "vm", "protocol": "tcp", "port": "389"},  {"service": "dns", "service\_type": "vm", "protocol": "udp", "port": "53"},  {"service": "ntp", "service\_type": "vm", "protocol": "tcp", "port": "123"}  ]  }  },  { "server": { "dev\_id": "S2" , "server\_name": "svr2" , "domain": "biasc.be", "ip-address": "10.2.3.2" ,  "os": "linux" , "server\_type": "vm" ,  "services": [  {"service": "flask", "service\_type": "vm", "protocol": "tcp", "port": "8089" },  {"service": "db" , "service\_type": "vm", "protocol": "tcp", "port": "1521" }  ]  }  },  { "server": { "dev\_id": "S3" , "server\_name": "svr3" , "domain": "biasc.be" , "ip-address": "10.2.3.3",  "os": "linux" , "server\_type": "vm" ,  "services": [  {"service": "dns" , "service\_type": "vm", "protocol": "tcp", "port": "8089" },  {"service": "ntp" , "service\_type": "vm", "protocol": "tcp", "port": "8089" },  {"service": "dhcp", "service\_type": "docker", "protocol": "udp", "port": "67" }  ]  }  }  ]  }  **Basic rules underlying the JSON structure**  ### RULES  inventory\_dict = {}  inventory\_list = [] #### [inventory\_dict]  **service\_dict** = {}  service\_list = [] #### [service\_dict]  **dev\_dict** = {}  dev\_list = [] #### [dev\_dict]  rack\_struc = {} #### {dev\_dict\_list}  rack\_struc["rack"] = []  #  *YANG leaves are comparable to a Python dict structure; YANG lists are comparable to an array (or list) in Python.* |