Network Automation Using Python and Google Sheets

Ievgen Kostiukevych
IP Media Technology Architect
European Broadcasting Union
| PP1 | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  | 16  | 17  | 18  | 19  | 20  | 21  | 22  | 23  | 24  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| PP2 | 28  | 28  | 87  | 88  | 88  | 20  | 31  | 32  | 32  | 34  | 35  | 36  | 37  | 38  | 39  | 41  | 42  | 42  | 44  | 44  | 45  | 46  | 47  | 48  |
| PP3 | 49  | 49  | 50  | 51  | 52  | 53  | 54  | 55  | 56  | 57  | 58  | 59  | 60  | 61  | 62  | 63  | 64  | 65  | 66  | 67  | 68  | 69  | 70  | 71  |
| PP4 | 72  | 73  | 74  | 75  | 76  | 77  | 78  | 79  | 80  | 81  | 82  | 83  | 84  | 85  | 86  | 87  | 88  | 89  | 90  | 91  | 92  | 93  | 94  | 95  |
| PP5 | 96  | 97  | 98  | 99  | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 |
| PP6 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 | 128 | 129 | 130 | 131 | 132 | 133 | 134 | 135 | 136 | 137 | 138 | 139 | 140 | 141 |
| PP7 | 142 | 143 | 144 | 145 | 146 | 147 | 148 | 149 | 150 | 151 | 152 | 153 | 154 | 155 | 156 | 157 | 158 | 159 | 160 | 161 | 162 | 163 | 164 |
| PP8 | 165 | 166 | 167 | 168 | 169 | 170 | 171 | 172 | 173 | 174 | 175 | 176 | 177 | 178 | 179 | 180 | 181 | 182 | 183 | 184 | 185 | 186 | 187 |
| PP9 | 188 | 189 | 190 | 191 | 192 | 193 | 194 | 195 | 196 | 197 | 198 | 199 | 200 | 201 | 202 | 203 | 204 | 205 | 206 | 207 | 208 | 209 | 210 |
| PP10| 211 | 212 | 213 | 214 | 215 | 216 | 217 | 218 | 219 | 220 | 221 | 222 | 223 | 224 | 225 | 226 | 227 | 228 | 229 | 230 | 231 | 232 | 233 |

| SY4 | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  | 16  | 17  | 18  | 19  | 20  | 21  | 22  | 23  | 24  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| SY2 | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31  | 32  | 33  | 34  | 35  | 36  | 37  | 38  | 39  | 40  | 41  | 42  | 43  | 44  | 45  | 46  |
| SY1 | 47  | 48  | 49  | 50  | 51  | 52  | 53  | 54  | 55  | 56  | 57  | 58  | 59  | 60  | 61  | 62  | 63  | 64  | 65  | 66  | 67  | 68  | 69  |

©Ievgen Kostiukevych, kostiukevych@ebu.ch, Special for NAB 2019 IP Showcase Theater
ARISTA-7020TR-LEAF-2.Wed-13:52:01#conf
% Invalid input
ARISTA-7020TR-LEAF-2.Wed-13:53:00(config-if-Et33)#desc Ward Beck AMS8-1AN
| E17 | 81446490 GMI- MONTRE-42 | connected | routed | a full | e full | 1000BASE-T |
| E18 | Riedel RIF-2358 | connected | 10 | a full | e full | 1000BASE-T |
| E19 | Riedel RSF 300 C2 | connected | 10 | a full | e full | 1000BASE-T |
| E20 | Shape TeX NESS 1700/1700 | connected | 10 | a full | e full | 1000BASE-T |
| E21 | Shape TeX NESS 1700/1700 | connected | 10 | a full | e full | 1000BASE-T |
| E22 | Studio Technology Model 5532-W2 | connected | 10 | a full | e full | 1000BASE-T |
| E23 | Studio Technology Model 5531-W2 | connected | 10 | a full | e full | 1000BASE-T |
| E24 | Studio Technology Model 5531-W2 | connected | 10 | a full | e full | 1000BASE-T |
| E25 | Studio Technology Model 5531-A8 | connected | 10 | a full | e full | 1000BASE-T |
| E26 | Ross Video EVO-MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E27 | Ross Video EVO-MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E28 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E29 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E30 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E31 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E32 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E33 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E34 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E35 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E36 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E37 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E38 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E39 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E40 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E41 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E42 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E43 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E44 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E45 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E46 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E47 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E48 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E49 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E50 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E51 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E52 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E53 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |
| E54 | Ross Video MHEV | connected | 10 | a full | e full | 1000BASE-T |

**EBU**

OPERATING EUROVISION AND EURORADIO

©Ievgen Kostiukevych, kostiukevych@ebu.ch, Special for NAB 2019 IP Showcase Theater
Idea

• A centralized, but dynamic and collaborative repository for often changed switch parameters.
• A centralized, but dynamic and collaborative view of troubleshooting data from the switch, the “show” commands.
• A real time update or periodic polling of both.
• Has to support Arista switches
Possible solutions

• CLI?
• Ansible?
• Python?
  – CLI (Telnet/SSH)?
  – JSON?
  – Native API?
Python

- telnetlib
- paramiko
- netmiko
- pyeapi
import getpass
import sys
import telnetlib

ip_address = "192.168.122.71"
user = 'admin'
password = 'admin'

tn = telnetlib.Telnet(ip_address)
tn.read_until("Username: ")
    tn.write(user + "\n")
tn.read_until("Password: ")
    tn.write(password + "\n")
    tn.write("enable\n")
    tn.write("super_secure_enable_pass\n")
    tn.write("show version\n")

print(tn.read_all())
```python
import paramiko
import time

ip_address = "192.168.122.72"
username = "admin"
password = "admin"

ssh_client = paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_client.connect(hostname=ip_address, username=username, password=password)

remote_connection = ssh_client.invoke_shell()
remote_connection.send("enable\n")
remote_connection.send("super_secure_enable_pass\n")
remote_connection.send("show version\n")
time.sleep(1)
output = remote_connection.recv(65535)
print(output)

ssh_client.close
```
```python
from netmiko import ConnectHandler

S1 = {
    'device_type': 'arista_eos',
    'ip': '192.168.122.72',
    'username': 'admin',
    'password': 'admin',
}

net_connect = ConnectHandler(**S1)
output = net_connect.send_command('show version')
print(output)
```
```python
import pyeapi
connect = pyeapi.client.connect(
    transport='https',
    host='192.168.22.72',
    username='admin',
    password='admin')

connectedSwitch = pyeapi.client.Node(connect)

connectedSwitch.enable('show version')
```
Simple eAPI request editor

This page lets you craft a single eAPI request, and explore the returned JSON. Note that this form creates real eAPI requests, so any configuration you perform will apply to this switch. Don’t know where to start? Read the API overview or try one of these examples: Check version, Create an ACL, Show virtual router, or View running-config.

API Endpoint: http://172.16.2.11/command-api

Commands:
1. show version

Version: 1

Format: 'json' ▲

Timesteps: false ▲

ID: EapiExplorer-1

Submit POST request

Request Viewer

```
1  {  
2    "jsonrpc": "2.0",  
3    "method": "runCmds",  
4    "params": [],  
5    "id": "EapiExplorer-1"  
6  }
```

Response Viewer

```
1  {  
2    "jsonrpc": "2.0",  
3    "result": [  
4      {  
5        "modelName": "vEOS",  
6        "internalVersion": "4.15.3F-281279; 4153F",  
7        "systemMacAddress": "00:0c:29:08:76:08",  
8        "serialNumber": "",  
9        "memTotal": 1897552,  
10        "memUtilization": 46127519.58,  
11        "memFree": 148496,  
12        "version": "4.15.3F",  
13        "architecture": "x86",  
14        "internalBuildId": "3545f255-b84f-41f0-b800-0ce5b0585346",  
15        "hardwareRevision": ""  
16      },  
17    ],  
18    "id": "EapiExplorer-1"  
19  }
```
# Google sheet access

def googleAuthorize():
    # Sets access to Google spreadsheets and Google drive
    scope = ['https://spreadsheets.google.com/feeds',
             'https://www.googleapis.com/auth/drive']
    # Imports the JSON access token and extracts credential
    credentials = ServiceAccountCredentials.from_json_keyfile_name(
                   args.api, scope)
    # Authorization with extracted credentials
    gc = gspread.authorize(credentials)
    logger.info(' - Google Cloud API authorization successful')
    return(gc)

# Initial authorization at program start
gc = googleAuthorize()

# Extracts data from the spreadsheet
switchConfigs = gc.open(config['spreadsheet'])
vlanList = switchConfigs.worksheet('Vlan List')
vlanPorts = switchConfigs.worksheet(
    'Interfaces VLAN Allocation and Descriptions')
interfacesMacTable = switchConfigs.worksheet('MAC addresses table')
interfacesStatusTable = switchConfigs.worksheet('Interfaces status table')
### Switches

<table>
<thead>
<tr>
<th>VLAN ID</th>
<th>VLAN Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>default</td>
</tr>
<tr>
<td>1</td>
<td>TEN</td>
</tr>
<tr>
<td>2</td>
<td>TWENTY</td>
</tr>
<tr>
<td>3</td>
<td>THIRTY</td>
</tr>
<tr>
<td>4</td>
<td>FORTY</td>
</tr>
<tr>
<td>5</td>
<td>FIFTY</td>
</tr>
<tr>
<td>6</td>
<td>SIXTY</td>
</tr>
<tr>
<td>7</td>
<td>SEVENTY</td>
</tr>
<tr>
<td>100</td>
<td>Management</td>
</tr>
<tr>
<td>200</td>
<td>Poe_1</td>
</tr>
<tr>
<td>210</td>
<td>Poe_2</td>
</tr>
<tr>
<td>220</td>
<td>Poe_3</td>
</tr>
<tr>
<td>230</td>
<td>Poe_4</td>
</tr>
<tr>
<td>240</td>
<td>Poe_5</td>
</tr>
</tbody>
</table>

### VLAN List

| Interface VLAN Allocation and Descriptions |

### VLAN

<table>
<thead>
<tr>
<th>Port</th>
<th>VLAN</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Device 1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Device 2</td>
</tr>
<tr>
<td>3</td>
<td>200</td>
<td>Device 3</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>Device 4</td>
</tr>
<tr>
<td>5</td>
<td>160</td>
<td>Device 5</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>Device 6</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>Device 7</td>
</tr>
<tr>
<td>8</td>
<td>220</td>
<td>Device 8</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>Device 9</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td>Device 10</td>
</tr>
<tr>
<td>11</td>
<td>30</td>
<td>Device 11</td>
</tr>
<tr>
<td>12</td>
<td>240</td>
<td>Device 12</td>
</tr>
</tbody>
</table>

©2019 Ievgen Kostiukevych, kostiukevych@ebu.ch, Special for NAB 2019 IP Showcase Theater
def setInterfaceDescriptions():
    # Reads interfaces description from the spreadsheet and writes to the switch
    logger.info('=================================================================')
    logger.info(' - Updating interfaces descriptions...')
    for port in vlanPorts.get_all_records():
        portDescription.set_description('Ethernet {}'.format(
            port['Port']), value=port['Description'])
    logger.info(' - Interfaces descriptions updated')

def createVlans():
    # Reads vlan list and names from the spreadsheet and writes to the switch
    logger.info('=================================================================')
    logger.info(' - Updating vlan table...')
    vlands = connectedSwitch.api('vlans')
    vland = False
    for vlan in vlandList.get_all_records():
        vlan.create_vlan('Vlan ID', name=vlan['Vlan Name'])
    logger.info(' - Vlan table updated')

def setInterfaceVlans():
    # Reads interfaces vlans list from the spreadsheet and writes to the switch
    logger.info('=================================================================')
    logger.info(' - Updating interface vlans...')
    currentVlans = portsVlans.get_all()
    for port in vlanPorts.get_all_records():
        try:
            if not str(port['Vlan']) == str(currentVlans['Ethernet']()['Port'])['access_vlan']:
                logger.warning(
                    ' - Modifying port {} - new access Vlan {}'('Ethernet {}').format(
                        port['Port'], port['Vlan']))
            else:
                pass
        except KeyError:
            pass
    logger.info(' - Interface vlans updated')
```python
def getInterfacesStatus():
    # Reads interfaces statuses and mac address table from the switch and writes to the spreadsheet
    logger.info('Updating interfaces status table...
    portInterfaces = connectedSwitch.enable('show Interfaces status')
    InterfacesStatus = portInterfaces[0]['result'][1]['interfaceStatuses']
    macAddrTable = connectedSwitch.enable('show mac address-table')
    unicastMacAddrTable = macAddrTable[0]['result'][1]['unicastTable'][1]['tableEntries']

    header = ['name', 'interfaceType', 'linkStatus', 'macAddress', 'entryType']
    # Define the cell range
    cellRange = InterfacesStatusTable.range('A2:Z56')
    # Sort by interface name
    interfaceList = [InterfaceInterface in InterfacesStatus.keys()]
    interfaceList.append(interface)
    interfaceList.sort()

    # Flatten the list of dicts into a list of values in order
    flattened_test_data = []
    for i in interfaceList:
        for entry in unicastMacAddrTable:
            for value in entry.values():
                if value == i:
                    InterfaceStatus[i].update(
                        {'macAddress': entry['macAddress']}
                        InterfaceStatus[i].update(
                            {'entryType': entry['entryType']}
                    )

    flattened_test_data.append(
        try:
            flattened_test_data.append(
                InterfaceStatus[i].update(
                    'vlanInformation'['vlansId'])
            except KeyError:
                flattened_test_data.append('N/A')
        for j in header:
            try:
                flattened_test_data.append(InterfacesTable[i][j])
            except KeyError:
                flattened_test_data.append('N/A')
    # Send flattened list to the cell range to be re-rendered as a table
    for i in range(len(cellRange):)
        try:
            cell.value = flattened_test_data[i]
        except IndexError:
            pass
    InterfacesStatusTable.update_cell('cellRange')
    logger.info('Interfaces status table updated')
```
<table>
<thead>
<tr>
<th>Interface</th>
<th>Vlan ID</th>
<th>autoNegotiateActive</th>
<th>bandwidth</th>
<th>description</th>
<th>duplex</th>
<th>interfaceType</th>
<th>lineProtocolStatus</th>
<th>linkStatus</th>
<th>MAC Address</th>
<th>Entry Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet1</td>
<td>1</td>
<td>FALSE</td>
<td>FALSE</td>
<td>0 Device 1</td>
<td>duplexFull EbraTestPhyPort up</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet10</td>
<td>30</td>
<td>FALSE</td>
<td>FALSE</td>
<td>0 Device 10</td>
<td>duplexFull EbraTestPhyPort up</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet11</td>
<td>20</td>
<td>FALSE</td>
<td>FALSE</td>
<td>0 Device 11</td>
<td>duplexFull EbraTestPhyPort up</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet12</td>
<td>240</td>
<td>FALSE</td>
<td>FALSE</td>
<td>0 Device 12</td>
<td>duplexFull EbraTestPhyPort up</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet2</td>
<td>20</td>
<td>FALSE</td>
<td>FALSE</td>
<td>0 Device 2</td>
<td>duplexFull EbraTestPhyPort up</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet3</td>
<td>200</td>
<td>FALSE</td>
<td>FALSE</td>
<td>0 Device 3</td>
<td>duplexFull EbraTestPhyPort up</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet4</td>
<td>20</td>
<td>FALSE</td>
<td>FALSE</td>
<td>0 Device 4</td>
<td>duplexFull EbraTestPhyPort up</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet5</td>
<td>100</td>
<td>FALSE</td>
<td>FALSE</td>
<td>0 Device 5</td>
<td>duplexFull EbraTestPhyPort up</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet6</td>
<td>20</td>
<td>FALSE</td>
<td>FALSE</td>
<td>0 Device 6</td>
<td>duplexFull EbraTestPhyPort up</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet7</td>
<td>10</td>
<td>FALSE</td>
<td>FALSE</td>
<td>0 Device 7</td>
<td>duplexFull EbraTestPhyPort up</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet8</td>
<td>220</td>
<td>FALSE</td>
<td>FALSE</td>
<td>0 Device 8</td>
<td>duplexFull EbraTestPhyPort up</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet9</td>
<td>20</td>
<td>FALSE</td>
<td>FALSE</td>
<td>0 Device 9</td>
<td>duplexFull EbraTestPhyPort up</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Management1</td>
<td>N/A</td>
<td>TRUE</td>
<td>TRUE</td>
<td>10000000000</td>
<td>MGMT</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
First - connect to the switch and check connection and eAPI

```python
connect = pyeapi.client.connect(
    transport='https', host=conf['ip'], username=conf['username'], password=conf['password'])
logger.info(' - Trying to connect to {} using provided username and password'.format(conf['ip']))
connectedSwitch = pyeapi.client.Node(connect)
hostname = connectedSwitch.enable('show hostname')
logger.info(' - Successfully connected to ' + hostname[0]['result']['hostname'])
version = connectedSwitch.enable('show version')
logger.info(pp.pprint(version[0]['result']))
```

Second - read initial data from the switch

```python
vlans = connectedSwitch.api('vlans')
portsVlans = connectedSwitch.api('switchports')
portsDescription = connectedSwitch.api('interfaces')
```

Start executing selected tasks when continious flag is not set

```python
if args.vlans_list and not args.continious:
    createVlans()

if args.interfaces_description and not args.continious:
    setInterfaceDescriptions()

if args.interfaces_vlans and not args.continuous:
    setInterfaceVlans()

if args.interfaces_status and not args.continuous:
    getInterfacesState()

if not args.continuous:
    logger.info('--------------------------------------------------------')
    logger.info(' - All tasks finished. Exiting...')
What about real time update?
```python
# Tasks repetition when continious flag is set
# Each task can be interrupted by the Google API exhaustion error. 1 minute wait is introduced.

try:
    while args.continuous:
        now = datetime.datetime.now().replace(microsecond=0)
        logger.info('task 1: %s' % str(now))
        logger.info('task 2: %s' % str(now))

        # Get hostname
        hostname = hostname[0]['result']['hostname']

        if args.vlans_list:
            try:
                createVlans()
            except gsSpread.exceptions.APIError:
                wait('api')
                gc = googleAuthorize()

        if args.interfaces_description:
            try:
                setInterfaceDescriptions()
            except gsSpread.exceptions.APIError:
                wait('api')
                gc = googleAuthorize()

        if args.interfaces_vlans:
            try:
                setInterfaceVlans()
            except gsSpread.exceptions.APIError:
                wait('api')
                gc = googleAuthorize()

        if args.interfaces_status:
            try:
                setInterfaceStatus()
            except gsSpread.exceptions.APIError:
                wait('api')
                gc = googleAuthorize()

except KeyboardInterrupt:
    logger.info('Stopped')
```

positional arguments:
config                  provide the config file
api                     provide the API json file
time                     amount of seconds to wait before repeating tasks,
                        default is 10 seconds

optional arguments:
-h, --help               show this help message and exit
-vL, --vlans_list       read vlans from the spreadsheet and create in the switch
-iD, --interfaces_description
                        read interfaces descriptions from the spreadsheet and update in the switch
-iS, --interfaces_status
                        read interfaces status from the switch and update in the spreadsheet
-iV, --interfaces_vlans
                        read interfaces vlans from the spreadsheet and update in the switch
-c, --continuous        repeat activated tasks after 10 seconds (default,
                        unless time is specified)
Useful links

- https://github.com/ktbyers/netmiko
- https://github.com/paramiko/paramiko
- https://docs.python.org/3.7/library/telnetlib.html
- https://www.gns3.com/
Thank You

Ievgen Kostiukevych
IP Media Technology Architect
European Broadcasting Union

kostiukevych@ebu.ch
+41 79 225 37 35