code_slideshow

May 20, 2020

0.0.1 Installation on Jupyter Notebook

```
import sys
!{sys.executable} -m pip install xmltodict
!{sys.executable} -m pip install netmiko
```

0.0.2 Installation on Terminal

```
pip install xmltodict
pip install netmiko
```

1 XML

[ ]:
```
# import the xmltodict library

import xmltodict
```

[ ]:
```
# Open the sample xml file and read it into variable
with open("xml/xml_example.xml") as f:
    xml_example = f.read()

# Print the raw XML data
print("type: ", type(xml_example))
print()
print(xml_example)
```

[ ]:
```

```

[ ]:
```
# Parse the XML into a Python dictionary

xml_dict = xmltodict.parse(xml_example)

print("type:", type(xml_dict))
print()
print(xml_dict)
print()
print(xml_dict.keys())
```

[ ]:

[ ]:
```
# Save the interface name into a variable using XML nodes as keys

int_name = xml_dict["interface"]["name"]

# Print the interface name
print(int_name)
```

[ ]:

[ ]:
```
# Change the IP address of the interface

xml_dict["interface"]["ipv4"]["address"]["ip"] = "192.168.0.2"

print(xml_dict)
```

[ ]:

[ ]:
```
# Revert to the XML string version of the dictionary


new_xml = xmltodict.unparse(xml_dict)
print(type(new_xml))
print()
print(new_xml)
```

[ ]:

2 JSON

[ ]:
```
# Import the json library

import json
```

[ ]:
```
# Open the sample json file and read it into variable

with open("json/json_example.json") as f:
    json_example = f.read()

# Print the raw json data
print("type:", type(json_example))
print()
print(json_example)
```

[ ]:

[ ]:
```
# Parse the json into a Python dictionary
json_dict = json.loads(json_example)

print(type(json_dict))
print()
print(json_dict)
```

[ ]:

[ ]:
```
# Save the interface name into a variable

int_name = json_dict["interface"]["name"]

# Print the interface name
print(int_name)
```

[ ]:

[ ]:
```
# Change the IP address of the interface

json_dict["interface"]["ipv4"]["address"][0]["ip"] = "192.168.0.2"
print(json_dict)
```

[ ]:

[ ]:
```
# Revert to the json string version of the dictionary

new_json = json.dumps(json_dict)
print(type(new_json))
print()
print(new_json)
```

[ ]:

3 YAML

[ ]:
```
# Import the yamltodict library

import yaml
```

[ ]:
```
# Open the sample yaml file and read it into variable

with open("yaml/yaml_example.yaml") as f:
    yaml_example = f.read()

# Print the raw yaml data
```

```
print(type(yaml_example))
print()
print(yaml_example)
```

[ ]:

[ ]:
```
# Parse the yaml into a Python dictionary

# yaml_dict = yaml.load(yaml_example) # .load is deprecated
yaml_dict = yaml.full_load(yaml_example)

print(type(yaml_dict))
print()
print(yaml_dict)
```

[ ]:

[ ]:
```
# Save the interface name into a variable
int_name = yaml_dict["interface"]["name"]

# Print the interface name
print(int_name)
```

[ ]:

[ ]:
```
# Change the IP address of the interface

yaml_dict["interface"]["ipv4"]["address"][0]["ip"] = "192.168.0.2"

print(yaml_dict)
```

[ ]:

[ ]:
```
# Revert to the yaml string version of the dictionary

new_yaml = yaml.dump(yaml_dict, default_flow_style=False)
print(type(new_yaml))
print()
print(new_yaml)
```

[ ]:

4 CSV

[ ]:
```
# Import the csv library

import csv
```

[ ]:
```
# Open the sample csv file and print it to screen

with open("csv/csv_example.csv") as f:
    print(f.read())
```

[ ]:

[ ]:
```
# Open the sample csv file, and create a csv.reader object

with open("csv/csv_example.csv") as f:
    csv_python = csv.reader(f)

    # Loop over each row in csv and leverage the data in code
    for row in csv_python:
        print("{device} is in {location} " \
              "and has IP {ip}.".format(
                  device = row[0],
                  location = row[2],
                  ip = row[1]
                  )
                )
```

[ ]:

[ ]:
```
#Adding a new Router

# Collect information from the user and save to variables
print("Let's add a new router.")
hostname = input("What is the hostname? ")
ip = input("What is the ip address? ")
location = input("What is the location? ")

# Create new list representing device
device = [hostname, ip, location]

# Open the csv file in "append" mode to add new device
with open("csv/csv_example.csv", "a") as f:
    # Create a csv.writer object from the file
    csv_writer = csv.writer(f)
    # Add new row based on new device
    csv_writer.writerow(device)
```

[ ]:

5 Netmiko

[ ]:
```
# import the netmiko library and regular expression api

from netmiko import ConnectHandler
import re
```

[ ]:
```
# create device credentials

device = {
             "address": "10.1.99.45",
             "ssh_port": 22,
             "username": "cisco",
             "password": "cisco"
          }
```

[ ]:
```
# Set device_type for netmiko

device["device_type"] = "cisco_ios"
```

[ ]:
```
# create a connection object

ch = ConnectHandler(ip = device["address"],
                    port = device["ssh_port"],
                    username = device["username"],
                    password = device["password"],
                    device_type = device["device_type"])

print(ch)
```

[ ]:
```
# inspect the connection handler object
print(dir(ch))
```

[ ]:
```
# retreive the configuration of an interface on the router

# Create a CLI command template
show_interface_config_temp = "show running-config interface {}"

# Create desired CLI command and send to device
command = show_interface_config_temp.format("GigabitEthernet0/0")
interface = ch.send_command(command)

# Print the raw command output to the screen
print(interface)
```

[ ]:

[ ]:
```
# Use regular expressions to parse the output for desired data

name = re.search(r'interface (.*)', interface).group(1)
description = re.search(r'description (.*)', interface).group(1)
ip_info = re.search(r'ip address (.*) (.*)', interface)
ip = ip_info.group(1)
netmask = ip_info.group(2)

# Print the info to the screen
print("The interface {name} has ip address {ip}/{mask}".format(
        name = name,
        ip = ip,
        mask = netmask,
        )
    )
```

[ ]:

5.1 Full code in in a script

[ ]:
```
# Set device_type for netmiko
device["device_type"] = "cisco_ios"

# Create a CLI command template
show_interface_config_temp = "show running-config interface {}"

# Open CLI connection to device
with ConnectHandler(ip = device["address"],
                    port = device["ssh_port"],
                    username = device["username"],
                    password = device["password"],
                    device_type = device["device_type"]) as ch:

    # Create desired CLI command and send to device
    command = show_interface_config_temp.format("GigabitEthernet0/0")
    interface = ch.send_command(command)

    # Print the raw command output to the screen
    print(interface)

    # Use regular expressions to parse the output for desired data
    name = re.search(r'interface (.*)', interface).group(1)
    description = re.search(r'description (.*)', interface).group(1)
    ip_info = re.search(r'ip address (.*) (.*)', interface)
    ip = ip_info.group(1)
```

```
    netmask = ip_info.group(2)

    # Print the info to the screen
    print("The interface {name} has ip address {ip}/{mask}".format(
            name = name,
            ip = ip,
            mask = netmask,
            )
        )
```

[ ]:

5.1.1 Retreive the running configuration of a Loopback interface

[ ]:
```
# Set device_type for netmiko
device["device_type"] = "cisco_ios"

# Create a CLI command template
show_interface_config_temp = "show running-config interface {}"

# Open CLI connection to device
with ConnectHandler(ip = device["address"],
                    port = device["ssh_port"],
                    username = device["username"],
                    password = device["password"],
                    device_type = device["device_type"]) as ch:

    # Create desired CLI command and send to device
    command = show_interface_config_temp.format("Loopback0")
    interface = ch.send_command(command)

    # Print the raw command output to the screen
    print(interface)

    # Use regular expressions to parse the output for desired data
    name = re.search(r'interface (.*)', interface).group(1)
    description = re.search(r'description (.*)', interface).group(1)
    ip_info = re.search(r'ip address (.*) (.*)', interface)
    ip = ip_info.group(1)
    netmask = ip_info.group(2)

    # Print the info to the screen
    print("The interface {name} has ip address {ip}/{mask}".format(
            name = name,
            ip = ip,
            mask = netmask,
            )
```

```
        )
```

[ ]:

5.2 Updating Configuration with CLI

5.2.1 Update IP Address on loopback interface

[ ]:
```
# Set device_type for netmiko
device["device_type"] = "cisco_ios"

# New Loopback Details
loopback = {"int_name": "Loopback0",
            "description": "Demo interface by CLI and netmiko",
            "ip": "192.168.103.1",
            "netmask": "255.255.255.0"}

# Create a CLI configuration
interface_config = [
    "interface {}".format(loopback["int_name"]),
    "description {}".format(loopback["description"]),
    "ip address {} {}".format(loopback["ip"], loopback["netmask"]),
    "no shut"
]

# Open CLI connection to device
with ConnectHandler(ip = device["address"],
                    port = device["ssh_port"],
                    username = device["username"],
                    password = device["password"],
                    device_type = device["device_type"]) as ch:

    # Send configuration to device
    output = ch.send_config_set(interface_config)

    # Print the raw command output to the screen
    print("The following configuration was sent: ")
    print(output)
```

[ ]:

5.3 Delete Loopback Interface

[ ]:
```
# Set device_type for netmiko
device["device_type"] = "cisco_ios"

# New Loopback Details
```

```
loopback = {"int_name": "Loopback0"}

# Create a CLI configuration
interface_config = [
    "no interface {}".format(loopback["int_name"])
]

# Open CLI connection to device
with ConnectHandler(ip = device["address"],
                    port = device["ssh_port"],
                    username = device["username"],
                    password = device["password"],
                    device_type = device["device_type"]) as ch:

    # Send configuration to device
    output = ch.send_config_set(interface_config)

    # Print the raw command output to the screen
    print("The following configuration was sent: ")
    print(output)
```

[ ]:

6 Restconf

[ ]:
```
# Make a dictionary of the device credentials

device = {
             "address": "ios-xe-mgmt.cisco.com",
             "netconf_port": 10000,
             "restconf_port": 9443,
             "ssh_port": 8181,
             "username": "root",
             "password": "D_Vay!_10&"
          }
```

[ ]:
```
# Import libraries

import requests, urllib3
import sys

# Disable Self-Signed Cert warning for demo
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
```

[ ]:
```
# Setup base variable for request

```

```
restconf_headers = {"Accept": "application/yang-data+json"}
restconf_base = "https://{ip}:{port}/restconf/data"
interface_url = restconf_base + "/ietf-interfaces:interfaces/
 ↪interface={int_name}"

# Create URL and send RESTCONF request to core1 for GigE1 Config
url = interface_url.format(ip = device["address"],
                           port = device["restconf_port"],
                           int_name = "GigabitEthernet1"
                          )
print("URL: {}\n".format(url))
```

[ ]:

[ ]:
```
r = requests.get(url,
        headers = restconf_headers,
        auth=(device["username"], device["password"]),
        verify=False)

# Print returned data (Even if an error is generated you will still get some␣
 ↪data)
print("GET DATA:")
print(type(r.text))
print()
print(r.text)
```

[ ]:

[ ]:
```
# Make a condition to check if request was successful or not

if r.status_code == 200:
    # Process JSON data into Python Dictionary and use
    interface = r.json()["ietf-interfaces:interface"]
    print("The interface {name} has ip address {ip}/{mask}".format(
            name = interface["name"],
            ip = interface["ietf-ip:ipv4"]["address"][0]["ip"],
            mask = interface["ietf-ip:ipv4"]["address"][0]["netmask"],
            )
        )
else:
    print("No interface {} found.".format("GigabitEthernet1"))
```

[ ]:

6.1 RESTCONF: Creating a New Loopback 1

[ ]:
```
# Setup base variable for request

restconf_headers = {"Accept": "application/yang-data+json"}
restconf_base = "https://{ip}:{port}/restconf/data"
interface_url = restconf_base + "/ietf-interfaces:interfaces/
 ↪interface={int_name}"

# Create URL and send RESTCONF request to core1 for Lo101 Config
url = interface_url.format(ip = device["address"],
                           port = device["restconf_port"],
                           int_name = "Loopback101"
                          )

print("URL: {}\n".format(url))

# make the request
response = requests.get(url,
        headers = restconf_headers,
        auth=(device["username"], device["password"]),
        verify=False)

# Print returned data
print("GET DATA:")
print(response.text)

if response.status_code == 200:
    # Process JSON data into Python Dictionary and use
    interface = r.json()["ietf-interfaces:interface"]
    print("The interface {name} has ip address {ip}/{mask}".format(
            name = interface["name"],
            ip = interface["ietf-ip:ipv4"]["address"][0]["ip"],
            mask = interface["ietf-ip:ipv4"]["address"][0]["netmask"],
            )
        )
else:
    print("No interface {} found.".format("Loopback101"))
```

[ ]:
```

```

[ ]:
```
# Setup base variable for request
restconf_headers = {"Accept": "application/yang-data+json",
                    "Content-Type": "application/yang-data+json"}
restconf_base = "https://{ip}:{port}/restconf/data"
interface_url = restconf_base + "/ietf-interfaces:interfaces/
 ↪interface={int_name}"
```

```
# New Loopback Details
loopback = {"name": "Loopback101",
            "description": "Demo interface by RESTCONF",
            "ip": "192.168.101.1",
            "netmask": "255.255.255.0"}

# Setup data body to create new loopback interface
data = {
    "ietf-interfaces:interface": {
        "name": loopback["name"],
        "description": loopback["description"],
        "type": "iana-if-type:softwareLoopback",
        "enabled": True,
        "ietf-ip:ipv4": {
            "address": [
                {
                    "ip": loopback["ip"],
                    "netmask": loopback["netmask"]
                }
            ]
        }
    }
}

# Create URL and send RESTCONF request to device
url = interface_url.format(ip = device["address"],
                           port = device["restconf_port"],
                           int_name = loopback["name"]
                          )
print("URL: {}\n".format(url))

r = requests.put(url,
        headers = restconf_headers,
        auth=(device["username"], device["password"]),
        json = data,
        verify=False)

# Print returned data
print("PUT Request Status Code: {}".format(r.status_code))
```

[ ]:

6.2 RESTCONF: Delete created loopback

[ ]:
```
# Disable Self-Signed Cert warning for demo
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# Setup base variable for request
restconf_headers = {"Accept": "application/yang-data+json"}
restconf_base = "https://{ip}:{port}/restconf/data"
interface_url = restconf_base + "/ietf-interfaces:interfaces/
 ↪interface={int_name}"

# Create URL and send RESTCONF request to core1 for GigE2 Config
url = interface_url.format(ip = device["address"],
                           port = device["restconf_port"],
                           int_name = "Loopback101"
                          )
print("URL: {}\n".format(url))

r = requests.delete(url,
        headers = restconf_headers,
        auth=(device["username"], device["password"]),
        verify=False)

# Print returned data
print("DELETE Request Status Code: {}".format(r.status_code))

# # Process JSON data into Python Dictionary and use
# interface = r.json()["ietf-interfaces:interface"]
# print("The interface {name} has ip address {ip}/{mask}".format(
#         name = interface["name"],
#         ip = interface["ietf-ip:ipv4"]["address"][0]["ip"],
#         mask = interface["ietf-ip:ipv4"]["address"][0]["netmask"],
#         )
#     )
```

[ ]:
```

```