# JPATH Engine

## Contents



## Evolution

As a part of Collectors Topology payload we keep on getting huge and complex JSON payloads. Life was difficult while navigating through the multilevel complex JSON structures. Unlike XML, JSON doesn't have the equivalent feature of XPATH. So, we thought of building some library which can imitate some basic behaviour of XPATH on XML for our Topology JSON, if not all. We call it JPATH. To start with, three features were on top of our mind:

1. Flatten the full JSON Into Key value pair.
2. Search for a key and get the JPATH of it.
3. Supply the JPATH and get the value for that JPATH.

## Description

We can get the description of the Library by using –help or -h command.

```
D:\vmware\BIBA\json_utility\v4>python jpath.py -h
usage: jpath.py [-h] [-m MODE] [-k KEYNAME] [-j JPATH] [-f FILE]

The script imitates some of the behaviour of Xpath as in xml for json.
    --> It supports absolute path as well as  wildcarded path.
     --> The script will run on getjpath mode ,getelement mode and flatten mode

optional arguments:
  -h, --help            show this help message and exit
  -m MODE, --mode MODE  This script will run in 3 mode 1."getjpath", 2."getelement" 3."flatten" mode
  -k KEYNAME, --keyname KEYNAME
                        keyname
  -j JPATH, --jpath JPATH
                        jpath
  -f FILE, --file FILE  path to valid json
```

# Use cases

## Flattening JSON

Suppose we have JSON file name as TopologyDiffChanges.json as shown below. It size is 315Kb.


..so on

.

To flatten this JSON run the following command.

```
$ python  jpath.py  -m flatten -f TopologyDiffChanges.json
```

```
u'payload.@class': u'com.vmware.iec.nova.json.transfer.TopologyDiffChanges',
u'payload.changes[0].addedEntities[0].metadata.config.alternateGuestName': u'',
u'payload.changes[0].addedEntities[0].metadata.config.annotation': u'',
u'payload.changes[0].addedEntities[0].metadata.config.bootOptions.bootDelay': 0,
u'payload.changes[0].addedEntities[0].metadata.config.bootOptions.bootRetryDelay': 10000,
u'payload.changes[0].addedEntities[0].metadata.config.bootOptions.bootRetryEnabled': False,
u'payload.changes[0].addedEntities[0].metadata.config.bootOptions.enterBIOSSetup': False,
u'payload.changes[0].addedEntities[0].metadata.config.changeTrackingEnabled': False,
u'payload.changes[0].addedEntities[0].metadata.config.changeVersion': u'2017-04-27T16:01:09.706004Z'
u'payload.changes[0].addedEntities[0].metadata.config.cpuAllocation.expandableReservation': False,
u'payload.changes[0].addedEntities[0].metadata.config.cpuAllocation.limit': -1,
u'payload.changes[0].addedEntities[0].metadata.config.cpuAllocation.reservation': 0,
u'payload.changes[0].addedEntities[0].metadata.config.cpuAllocation.shares.level': u'normal',
u'payload.changes[0].addedEntities[0].metadata.config.cpuAllocation.shares.shares': 2000,
u'payload.changes[0].addedEntities[0].metadata.config.cpuHotAddEnabled': False,
u'payload.changes[0].addedEntities[0].metadata.config.cpuHotRemoveEnabled': False,
```

## Search for the key and get JPATH

### Scenarios 1

In this case we are searching for key – “nodeId”. To do this type follwing command-

**Command-**

python jpath.py

-f output_1497022158193_Topology.json
-m getJPATH
-k nodeId

**Output-**

```
:\vmware\BIBA\json_utility>python jpathengine.py -f D:\vmware\BIBA\skyline\NSX_output\output_1497022158193_Topology.json -m getjpath -k nodeId

he path till 'nodeId'in the JSON is-
>> payload.entities[0].metadata.virtualMachineInfo.nodeId
>> payload.entities[0].metadata.clusterInfo.nodeId
>> payload.entities[0].metadata.resourcePoolInfo.nodeId
>> payload.entities[0].metadata.datastoreInfo.nodeId
>> payload.entities[0].metadata.hostInfo.nodeId
>> payload.entities[1].metadata.virtualMachineInfo.nodeId
>> payload.entities[1].metadata.clusterInfo.nodeId
>> payload.entities[1].metadata.resourcePoolInfo.nodeId
>> payload.entities[1].metadata.datastoreInfo.nodeId
>> payload.entities[1].metadata.hostInfo.nodeId
```

## Scenarios 2

From the use case 1 you got the various JPATH for “nodeId”. Which will give you the information about the levels at which key – “nodeId” is present. Suppose I want to get the “node id” which is present at – “**payload.entities[1].metadata.virtualMachineInfo.nodeId”.** To do this we should give following command.

**Command-**

python Jpath.py
-f output_1497022158193_Topology.json
-m getelement
-j payload.entities[1].metadata.virtualMachineInfo.nodeId

**Output**

**[**u'193e3fc0-372f-405f-a717-2dc937186db4'**]**

## Scenarios 3

If we want to give multiple JPATH at a same time. Just pass them semicolon delimited.

**Command-**

python Jpath.py
-f output_1497022158193_Topology.json
-m getelement

-j payload.entities[0].metadata.virtualMachineInfo.nodeId**;**
payload.entities[1].metadata.virtualMachineInfo.nodeId

**output**

**[**u'193e3fc0-372f-405f-a717-2dc937186db4', u'193e3fc0-372f-405f-a717-2dc937186db4'**]**

## Scenarios 4

Using range instead of multiple JPATH.

**Command**

python Jpath.py
-f output_1497022158193_Topology.json
-m getelement
-j payload.entities**[0to2].**metadata.virtualMachineInfo.nodeId

**output**

**[**u'193e3fc0-372f-405f-a717-2dc937186db4', u'193e3fc0-372f-405f-a717-2dc937186db4'**]**

## Scenarios 5

Using wildcarded * instead for range or multiple JPATH

**Command**

python Jpath.py
-f output_1497022158193_Topology.json
-m getelement
-j payload.entities**[*].**metadata.virtualMachineInfo.nodeId

**output**

**[**u'193e3fc0-372f-405f-a717-2dc937186db4', u'193e3fc0-372f-405f-a717-2dc937186db4'**]**