JSON Support for Junos OS

1 Introduction:

Java Script Object Notation alias JSON a light weight data exchange format is being extensively used for data exchange between web application and servers. With the JUNOS Automation tools extending its wings to Python language support and REST feature, JSON data format needs to be supported by JUNOS in addition to the existing XML to make the data exchange simpler and faster.

JSON has an edge over XML when it comes to data interchange and some of its advantages over XML are listed below

- Much simpler and far lesser content which makes it fat free when compared with XML.
- The JSON grammar maps more directly onto the data structures used in modern programming languages like Python and Java script thereby makes it easier to process using the native language support.
- Easier to read and interpret for both human and systems.

JUNOS Support both the configuration and the operational data in JSON format.

2 JSON Representation and Data types:

2.1 Representation

JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.

  Example : “Name” : “Value”

- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

  Example : { "Alplhabhet_List": ["a", "b", "c", "d"] }

2.2 Data Types

Number

```
{ "myNum": 123.456 }
```

A series of numbers; decimals ok; double-precision floating-point format.

String

```
{ "myString": "abcdef" }
```

A series of characters (letters, numbers, or symbols); double-quoted UTF-8 with backslash escaping.

Boolean

```
{"myBool": true }
```

True or false.

Array

```
{ "myArray": [ "a", "b", "c", "d" ] }
```

Sequence of comma-separated values (any data type); enclosed in square brackets.

Object

```
{ "myObject": { "id": 7 } };
```

Unordered collection of comma-separated key/value pairs; enclosed in curly braces; properties (keys) are distinct strings.

Null

```
{ "myNull": null }
```

Variable with null (empty) value.

3 Configuration in JSON

The configuration of JUNOS can be represented in JSON. This section provides some of the basic configuration models in XML and their equivalent JSON format.

3.1 Leaf nodes attributes in JSON

The leaf nodes are just emitted as key value pairs in JSON.

XML :

```
<system>
            <host-name>Hercules</host-name>
 </system>
```

JSON :

```
{
   "system": {
      "host-name": " Hercules"
      }
}
```

3.2 Leaf nodes attributes with meta-data in JSON

The metadata of leaf nodes are emitted as separate entry with the key starting with "@" character followed by the name of the corresponding leaf node.

XML :

```
<system>
            <host-name>Hercules</host-name>
 </system>
```

JSON :

```
{
   "system": {
        "host-name": " Hercules"
   }
    "@host-name" : {
        "inactive" : true
    }
}
```

3.3 Leaf node with empty data

NULL is used to represent the empty data and meta-data will be represented in the same fashion as leaf node.

XML :

```
<system>
    <commit>
        <persist-groups-inheritance/>
    </commit>
</system>
```

JSON :

```
"system" : {
       "commit" :
       {
           "persist-groups-inheritance" : [null]
            "@persist-groups-inheritance" : {
                "inactive" : true
            }
```

```
        }
    }
```

3.4 Simple Objects with meta-data

The metadata of the simple objects are emitted inside the JSON objects as separate JSON element with the key as "@" character followed by the individual meta-data and their values.

XML :

```
    <system inactive="inactive">
        <host-name>Hercules</host-name>
    </system>
```

JSON :

```
    {
        "system" : {
            "@" : {
                "inactive" : true
            },
            "host-name" : "Hercules"
        }
    }
```

3.5 Containers Nodes

Container Nodes are emitted as JSON Array. The meta-deta of the individual elements inside the container are represented inside the JSON element as a separate element with key as "@".

XML :

```
<interfaces>
      <interface inactive="inactive">
          <name>ge-0/0/0</name>
          <description>foo</description>
      </interface>
       <interface>
            <name>ge-0/0/1</name>
            <description>bar</description>
      </interface>
</interfaces>
```

JSON :

```
        "interfaces" : {
              "interface" : [
              {
                  "@" : {
                      "inactive" : true
                  },
                  "name" : "ge-0/0/0",
                  "description" : "foo"
              },
              {
                  "name" : "ge-0/0/1",
                  "description" : "bar"
              }
              ]
        }
```

3.6 Leaf List in JSON

The Leaf List in JSON is also emitted as JSON Array.

XML :

```
    <system>
        <authentication-order>password</authentication-order>
        <authentication-order>tacplus</authentication-order>
    </system>
```

JSON :

```
    "system": {
       "authentication-order" :  [ “password” , “tacplus”]
     }
```

4 Display/Loading of JUNOS Configuration in JSON Format

The configuration of the JUNOS can be emitted and loaded both from CLI and Netconf / Junoscript. The configuration emitted can be parsed by any of the third party JSON parser. The parsed JSON data can be interpreted using any of the programming languages to access the configuration data and its values. The configuration can also be modified programmatically. The Modified configuration data can again be loaded to the JUNOS device using Netconf RPC.

4.1 CLI

4.1.1 Displaying Configuration in JSON

The “show configuration” command is used to emit the configuration. A new option called "json" will be added under "| display" option for emitting the configuration in JSON format.

The 'show configuration | display json" will display the configuration in JSON.

```
Router> show configuration | display ?
Possible completions:
json                  Show output in JSON format
```

Example:

```
Router# show | display json
{
    "configuration" : {
        "system" : {
            "services" : {
                "ftp" : null,
                "ssh" : {
                    "root-login" : "allow"
                },
                "telnet" : null,
                "xnm-clear-text" : null,
              }
        }
    }
}
```

4.1.2 Load JSON Configuration

The Configuration in JSON can be loaded using the following commands for various modes of operations.

load json merge <filename/terminal> <relative>

load json override <filename/terminal>

load json update <filename/terminal**>**

Example :

```
      Router# load override json config_file_json.txt
```

4.2 NETCONF/JUNOSCRIPT

4.2.1 Displaying Configuration in JSON

The "get-configuration" RPC is used is used to emit the configuration. The JSON format can be specified as a XML attribute in the xml tag of get-configuration.

```
      <rpc>  <get-configuration format="json"/> </rpc>
```

Example :

```
<rpc>  <get-configuration format="json"/> </rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
{
    "configuration" : {
        "version" : "15.1I20150610_1111_balasank [balasank]",
        "system" : {
            "services" : {
                "ftp" : null,
                "ssh" : {
                    "root-login" : "allow"
                },
                "telnet" : null,
                "xnm-clear-text" : null,
                "netconf" : {
                    "ssh" : {
                        "port" : "830"
                    }
                },
                "web-management" : {
                    "http" : null
                }
            }
        }
    }
}
</rpc-reply>
```

4.2.2 Load JSON Configuration

The Configuration in JSON can be loaded using the load-configuration RPC

```
         <rpc>
                      <load-configuration format="json"
operation=[merge|override|update]>
                        <configuration-json>
                            [Config in json]
                         </configuration-json>
                       </load-configuration>
            </rpc>
```

Example :

```
 <rpc>
<load-configuration format="json">
<configuration-json>
{
    "configuration" : {
        "system" : {
            "services" : {
                "ftp" : null,
                "ssh" : {
                    "root-login" : "allow"
                },
                "telnet" : null,
                "xnm-clear-text" : null,
                "netconf" : {
                    "ssh" : {
                        "port" : "834"
                    }
                },
                "web-management" : {
                    "http" : null
                }
            }
        }
    }
}
</configuration-json>
</load-configuration>
  </rpc>
```

5 JSON Data for Operational Commands

Junos emits the data for operational commands natively in XML. The XML data then gets converted to text depending upon the format defined in Output Definition Language (ODL). In order to emit the operational data in JSON, the XML data has to be converted to JSON.

Some of the convention for the XML to JSON format for operational commands is mentioned below. The Operation data can be viewed in JSON using ‘display json’ or “format= json” as already mentioned for configuration commands.

5.1 XML Leaf node

The value of the leaf node will be emitted with "data" as name. Value for the xml toggle will be represented with "null".

XML :

```
    <name>fxp0</name>
```

JSON:

```
    {
      "name" : [
        {
          "data" : "fxp0",
        }
      ]
    }
```

5.2 Simple XML leaf node with attributes

XML :

```
  <admin-status junos:format="Enabled">up</admin-status>
```

JSON :

```
    {
      "admin-status" : [
        {
          "data" : "up",
          "attributes" : {"junos:format" : "Enabled" }
        }
      ]
    }
```

5.3 Simple leaf node with empty data

XML:

```
  <ifdf-present/>
```

JSON:

```
    {
```

```
      "ifdf-present" : [
        {
          "data" : null,
        }
      ]
    }
```

5.4 Simple XML Element

XML

```
 <physical-interface>
      ....
  </physical-interface>
```

JSON:

```
    {
      "physical-interface" : [
        {
          ....
        }
      ]
    }
```

Example :

XML :

```
 <rpc-reply
xmlns:junos="http://xml.juniper.net/junos/13.3I0/junos">
       <interface-information
xmlns="http://xml.juniper.net/junos/13.3I0/junos-interface"
junos:style="normal">
           <physical-interface>
               <name>cbp0</name>
               <admin-status junos:format="Enabled">up</admin-
status>
           </physical-interface>
           <physical-interface>
               <name>fxp0</name>
               <oper-status>up</oper-status>
               <if-device-flags>
               <ifdf-present/>
               <ifdf-running/>
               </if-device-flags>
           </physical-interface>
       </interface-information>
```

```
   </rpc-reply>
```

JSON:

```
        "interface-information" : [
        {
            "__attributes" : { "xmlns" :
"http://xml.juniper.net/junos/13.3I0/junos-interface" ,
"junos:style" : "normal" },
            "physical-interface" : [
            {
                "name" : [
                {
                    "data" : "cbp0",
                }
                ],
                "admin-status" : [
                {
                    "data" : "up",
                    "attributes" : {"junos:Format : "Enabled" }
                }
                ]
            },
            {
                "name" : [
                {
                    "data" : "fxp0",
                }
                ],
                "oper-status" : [
                {
                    "data" : "up",
                }
                ],
                "if-device-flags" : [
                {
                    "ifdf-present" : [
                    {
                        "data" : null,
                    }
                    ],
                    "ifdf-running" : [
                    {
                        "data" : null,
                    }
                    ]
               }
               ]
```

```
                }
                ]
        }
        ]
```