



Model-Driven Command-Line Interface

This section shows the CLI commands that are based on YANG data models and can be used on the router console.

- [Model-Driven CLI to Display Data Model Structure, on page 1](#)
- [Model-Driven CLI to Display Running Configuration in XML and JSON Formats, on page 5](#)

Model-Driven CLI to Display Data Model Structure

Table 1: Feature History Table

Feature Name	Release Information	Description
Model-driven CLI to Show YANG Operational Data	Release 7.3.2	<p>This feature enables you to use a traditional CLI command to display YANG data model structures on the router console and also obtain operational data from the router in JSON or XML formats. The functionality helps you transition smoothly between CLI and YANG models, easing data retrieval from your router and network.</p> <p>This feature introduces the show yang operational command.</p>

Cisco IOS XR Software provides a rich set of show commands and data models to access data from the router and network. The show commands present unstructured data, whereas data models are structured data that can be encoded in XML or JSON formats. However, both the access points do not always present the same view. Network operators who work on show commands face challenges with adopting the data models when transitioning to programmatic interfaces.

With this feature, these adoption challenges are overcome using **show yang operational** command that is driven by data models. The command uses the data model as the base to display the structured data using traditional CLI command. Using this command, you can simplify parsing scripts via XML and JSON formats.

A data model has a structured hierarchy: model, module, container, and leaf. The following example shows the structure of `ietf-interfaces.yang` data model:

```

ietf-interfaces.yang
module: ietf-interfaces
+--rw interfaces
| +--rw interface* [name]
|   +--rw name                string
|   +--rw description?        string
|   +--rw type                 identityref
|   +--rw enabled?            boolean
|   +--rw link-up-down-trap-enable? enumeration {if-mib}?
+--ro interfaces-state
  +--ro interface* [name]
  +--ro name                string
  +--ro type                 identityref
  +--ro admin-status        enumeration {if-mib}

```

In the example, the hierarchy of the data model is as follows:

- Model—ietf-interfaces.yang
- Module—ietf-interfaces
- Container—interfaces, interface-state
- Node—interface* [name]
- Leaf—name, description, type, enabled, link-up-down-trap-enable, admin-status

You can use the **show yang operational** command to navigate to the leaf level as you do in a data model.

The image shows a mapping between CLI and data model, and how the structured data is displayed on the console.

```

14 RP/0/RSP0/CPU0:vk4# show yang ?
15
16 aaa
17 acl
18 arp
19 ...
20 inventory
21 ...
22
23
24 RP/0/RSP0/CPU0:vk4# show yang inventory ?
25 entities      Entities Table
26 racks         Rack Table
27 xml           Output in XML format.
28 |             Output Modifiers
29 |<cr>
30
31
32 RP/0/RSP0/CPU0:vk4# show yang inventory entities ?
33 entity        Actual entity name
34
35
36
37 RP/0/RSP0/CPU0:vk4# show yang inventory entities
38 [Cisco-IOS-XR-invmgr-oper inventory entitiis]
39 entity/name=Rack 0
40 attributes
41   inv-basic-bag
42     description: ASR-9904 AC Chassis
43     vendor-type: 1.3.6.1.4.1.9.12.3.1.3.1301
44     name: Rack 0
45     hardware-revision: V01
46     software-revision: 7.2.1.24I
47     serial-number: FOX2012GA1J
48     manufacturer-name: CISCO SYSTEMS, INC
49     model-name: ASR-9904-AC
50     is-field-replaceable-unit: true
51     composite-class-code: 65536
52     unrecognized-fru: false
53     unique-id: 8384513
54   inv-asset-bag
55     part-number: E0
56     manufacturer-assembly-number: 68-4854-01
57     manufacturer-assembly-revision: E0
58     manufacturer-common-language-equipment-identifier: IPWd00BARA
59
60

```

```

7 module: Cisco-IOS-XR-invmgr-oper
8   +--ro inventory
9     +--ro entities
10      ...
11      +--ro racks
12      ...
13
14
15
16
17
18 Yang module: Cisco-IOS-XR-invmgr-oper
19   +--ro inventory
20     +--ro entity* [name]
21       +--ro attributes
22         +--ro inv-basic-bag
23           +--ro description?          string
24           +--ro vendor-type?         string
25           +--ro name?                 string
26           +--ro hardware-revision?   string
27           +--ro firmware-revision?   string
28           +--ro software-revision?   string
29           +--ro chip-hardware-revision? string
30           +--ro serial-number?       string
31           +--ro manufacturer-name?   string
32           +--ro model-name?          string
33           +--ro asset-id-str?        int32
34           +--ro asset-identification? int32
35           +--ro is-field-replaceable-unit? boolean
36           +--ro manufacturer-asset-tags? int32
37           +--ro composite-class-code? int32
38           +--ro memory-size?        int32
39           +--ro environmental-monitor-path? string
40           +--ro alias?               string
41           +--ro group-flag?          boolean
42           +--ro new-deviation-number? int32
43           +--ro physical-layer-interface-module-type? int32
44           +--ro unrecognized-fru?    boolean
45           +--ro redundancystate?     int32
46           +--ro ceport?              boolean
47           +--ro xr-scoped?           boolean
48           +--ro unique-id?           int32
49         +--ro inv-asset-bag
50           +--ro part-number?         string
51           +--ro manufacturer-assembly-number? string
52           +--ro manufacturer-assembly-revision? string
53           +--ro manufacturer-firmware-identifier? string
54           +--ro manufacturer-software-identifier? string
55           +--ro manufacturer-common-language-equipment-identifier? string
56           +--ro original-equipment-manufacturer-string? string

```

The table shows various queries that can be used to navigate through the hierarchy of a data model using the CLI command. The queries are demonstrated using `Cisco-IOS-XR-interfaces-oper.yang` data model as an example.

Operational Query	Description
Search specific top-level nodes	<p>Search and produce the output of keywords from top-level nodes.</p> <pre>Router#show yang operational</pre> <pre>Router#show yang operational include <component></pre> <p>The following example shows the search result for interfaces:</p> <pre>Router#show yang operational include interface Wed Jul 7 00:02:37.982 PDT drivers-media-eth-oper:ethernet-interface ifmgr-oper:interface-dampening ifmgr-oper:interface-properties interface-cem-oper:cem l2vpn-oper:generic-interface-list-v2 pfi-im-cmd-oper:interfaces</pre>
All the instances of the container	<p>Lists all the models at the root level container and its container name.</p> <pre>Router#show yang operational ?</pre> <p>You can also see the containers for a partially typed keyword. For example, keyword search for <code>mpls-</code> displays all the containers with <code>mpls</code> :</p> <pre>Router#show yang operational mpls- mpls-io-oper-mpls-ea mpls-io-oper-mpls-ma mpls-ldp-mlldp-oper:mpls-mlldp mpls-lsd-oper:mpls-lsd mpls-lsp-oper:mpls-lsd-nodes mpls-ldp-mlldp-oper:mpls-mlldp mpls-vpn-oper:l3vpn mpls-te-oper:mpls-tp mpls-te-oper:mpls-te</pre> <p>View the container data. The output of the command is in-line with the structure of the data model.</p> <pre>Router#show yang operational mpls-static-oper:mpls-static Request datatree: filter mpls-static (ka) { "Cisco-IOS-XR-mpls-static-oper:mpls-static": { "vrfs": { "vrf": [{ "vrf-name": "default" }] }, "summary": { "lsp-count": 0, "label-count": 0, "label-error-count": 0, "label-discrepancy-count": 0, "vrf-count": 1, "active-vrf-count": 1, "interface-count": 0, "interface-forward-reference-count": 0, "lsd-connected": true, "ribv4-connected": false, "ribv6-connected": false } } }</pre>

Operational Query	Description
All the nodes of the container	<pre>Router#show yang operational mpls-static-oper:mpls-static ? JSON Output in JSON format XML Output in XML format local-labels summary vrfs Output Modifiers <cr></pre> <p>Output in JSON Format:</p> <pre>Router#show yang operational man-netconf-oper:netconf-yang clients JSON Mon Sep 27 11:38:27.158 PST Request datatree: filter netconf-yang (ka) clients { "Cisco-IOS-XR-man-netconf-oper:netconf-yang": { "clients": { "client": [{ "session-id": "1396267443", "version": "1.1", "connect-time": "52436839", "last-op-time": "1545", "last-op-type": "get", "locked": "No" }] } } }</pre> <p>Output in XML Format:</p> <pre>Router#show yang operational man-netconf-oper:netconf-yang clients XML Mon Sep 27 11:38:34.218 PST Request datatree: filter netconf-yang (ka) clients <netconf-yang xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-man-netconf-oper"> <clients> <client> <session-id>1396267443</session-id> <version>1.1</version> <connect-time>52443884</connect-time> <last-op-time>1545</last-op-time> <last-op-type>get</last-op-type> <locked>No</locked> </client> </clients> </netconf-yang></pre>

Operational Query	Description
Navigate until the last leaf level	<pre>Router#show yang operational mpls-static-oper:mpls-static summary ? JSON Output in JSON format XML Output in XML format active-vrf-count im-connected interface-count interface-forward-reference-count mpls-enabled-interface-count vrf-count Output Modifiers <cr></pre> <p>View data specific to the leaf value. The <code>read only (ro)</code> leaves in a YANG model are considered as the state data (operational).</p> <pre>Router#show yang operational mpls-static-oper:mpls-static summary active-vrf-count Request datatree: filter mpls-static (ka) summary active-vrf-count { "Cisco-IOS-XR-mpls-static-oper:mpls-static": { "summary": { "active-vrf-count": [] } } }</pre>

Model-Driven CLI to Display Running Configuration in XML and JSON Formats

Table 2: Feature History Table

Feature Name	Release Information	Description
Model-driven CLI to Display Running Configuration in XML and JSON Formats	Release 7.3.2	<p>This feature enables you to display the configuration data for Cisco IOS XR platforms in both JSON and XML formats.</p> <p>This feature introduces the show run [xml json] command.</p>

The **show run | [xml | json]** command uses native, OpenConfig and unified models to retrieve and display data.

Use the following variations of the command to generate output:

- **show run | [xml | json]**—Shows configuration in YANG XML or JSON tree.
- **show run | [xml | json] openconfig**—Shows configuration in OpenConfig YANG XML tree.

- **show run | [xml | json] unified**—Shows configuration in unified model YANG XML tree.
- **show run component | [xml | json]**—Shows configuration in YANG XML or JSON tree for the top-level component. For example, **show run interface | xml**
- **show run component | [xml | json] unified**—Shows configuration in unified model YANG XML or JSON tree for the top-level component. For example, **show run interface | json unified**
- **show run component subcomponent | [xml | json]**—Shows configuration in YANG XML or JSON tree for the granular-level component. For example, **show run router bgp 12 neighbor 12.12.12.12 | xml**
- **show run component subcomponent | [xml | json] unified**—Shows configuration in unified model YANG XML or JSON tree for the granular-level component. For example, **show run router bgp 12 neighbor 12.12.12.12 | json unified**

XML Output

```
Router#show run | xml
Building configuration...
<data>
  <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
    <interface-configuration>
      <active>act</active>
      <interface-name>GigabitEthernet0/0/0/0</interface-name>
      <shutdown></shutdown>
    </interface-configuration>
    <interface-configuration>
      <active>act</active>
      <interface-name>GigabitEthernet0/0/0/1</interface-name>
      <shutdown></shutdown>
    </interface-configuration>
    <interface-configuration>
      <active>act</active>
      <interface-name>GigabitEthernet0/0/0/2</interface-name>
      <shutdown></shutdown>
    </interface-configuration>
  </interface-configurations>
  <interfaces xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-um-interface-cfg">
    <interface>
      <interface-name>GigabitEthernet0/0/0/0</interface-name>
      <shutdown/>
    </interface>
    <interface>
      <interface-name>GigabitEthernet0/0/0/1</interface-name>
      <shutdown/>
    </interface>
    <interface>
      <interface-name>GigabitEthernet0/0/0/2</interface-name>
      <shutdown/>
    </interface>
  </interfaces>
</data>
```

JSON Output

```
Router#show run | json
Building configuration...
{
  "data": {
    "Cisco-IOS-XR-ifmgr-cfg:interface-configurations": {
      "interface-configuration": [
        {
          "active": "act",
```

```

    "interface-name": "GigabitEthernet0/0/0/0",
    "shutdown": [
      null
    ]
  },
  {
    "active": "act",
    "interface-name": "GigabitEthernet0/0/0/1",
    "shutdown": [
      null
    ]
  },
  {
    "active": "act",
    "interface-name": "GigabitEthernet0/0/0/2",
    "shutdown": [
      null
    ]
  }
],
"Cisco-IOS-XR-man-netconf-cfg:netconf-yang": {
  "agent": {
    "ssh": true
  }
},
}

```

Granular-Level Component Output

```
Router#sh run router bgp 12 neighbor 12.12.12.12 | json unified
```

```

{
  "data": {
    "Cisco-IOS-XR-um-router-bgp-cfg:router": {
      "bgp": {
        "as": [
          {
            "as-number": 12,
            "neighbors": {
              "neighbor": [
                {
                  "neighbor-address": "12.12.12.12",
                  "remote-as": 12,
                  "address-families": {
                    "address-family": [
                      {
                        "af-name": "ipv4-unicast"
                      }
                    ]
                  }
                }
              ]
            }
          }
        ]
      }
    }
  }
}

```

Unified Model Output

```
Router#sh run router bgp 12 | xml unified
```

```

<data>
  <router xmlns=http://cisco.com/ns/yang/Cisco-IOS-XR-um-router-bgp-cfg>
    <bgp>

```

```
<as>
  <as-number>12</as-number>
  <bgp>
    <router-id>1.1.1.1</router-id>
  </bgp>
  <address-families>
    <address-family>
      <af-name>ipv4-unicast</af-name>
    </address-family>
  </address-families>
  <neighbors>
    <neighbor>
      <neighbor-address>12.12.12.12</neighbor-address>
      <remote-as>12</remote-as>
      <address-families>
        <address-family>
          <af-name>ipv4-unicast</af-name>
        </address-family>
      </address-families>
    </neighbor>
  </neighbors>
</as>
</bgp>
</router>
</data>
```