Model Driven Command Line Interface

Model-driven CLI features for data model visualization

A model-driven CLI feature is a programmability capability in Cisco IOS XR software that

- enables users to display YANG data model structures and operational data directly via CLI commands,
- provides configuration and operational output in both XML and JSON formats for easier parsing and automation, and
- introduces specialized CLI commands that help transition between traditional CLI and model-driven approaches.

Model-driven CLI features are designed to bridge the gap between legacy command-line workflows and modern, programmatic network management. By leveraging YANG-based models, these features allow network operators to view, query, and retrieve data in structured formats, streamlining the integration with external tools and automation systems.

Table 1: Feature History Table

Feature Name	Release Information	Description
Model-driven CLI to Show YANG Operational Data	Release 7.3.2	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. This feature introduces the **show yang operational** command.

Cisco IOS XR Software provides a rich set of show commands and data models to access data from the router and network. Show commands present unstructured data, while data models provide structured data in XML or JSON formats; however, both methods can show different views. The model-driven CLI feature addresses

adoption challenges by allowing operators to use the familiar CLI while accessing structured, model-driven output. This enables easier integration with parsing scripts and automation tools.

Structure of data model

The structure of a data model organizes configuration and operational data in a hierarchical format. Each data model consists of these primary components:

- Model: The highest-level YANG file that defines a data schema e.g., `ietf-interfaces.yang`.
- Module: A logical collection of related definitions within the model e.g., `ietf-interfaces`.
- Container: A grouping node that organizes related nodes together e.g., `interfaces`, `interfaces-state`.
- List/Node: An array or list element within a container, often representing multiple items e.g., `interface* [name]`.
- Leaf: A single value node containing configuration or state data e.g., `name`, `description`, `type`, `enabled`, `admin-status`.

Table 2: Summary table of data model structure:

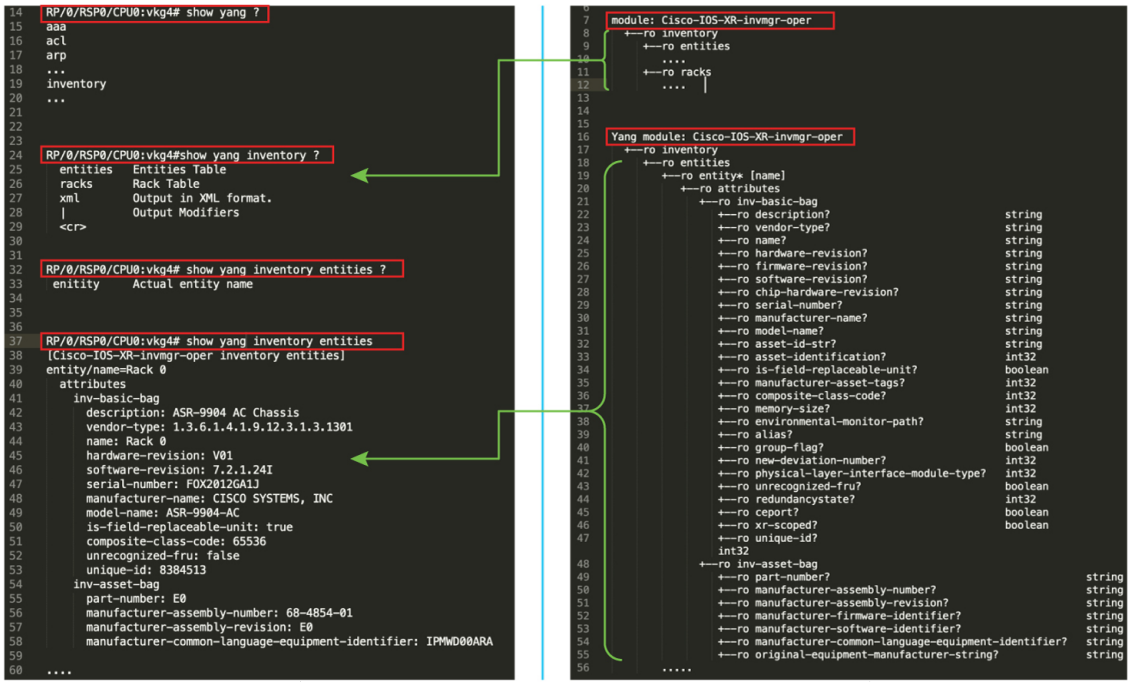
Component	Description	Example
Model	YANG file containing schema definitions	`ietf-interfaces.yang`
Module	Group of related schema nodes	`ietf-interfaces`
Container	Grouping node for related lists and leaves	`interfaces`, `interfaces-state`
List/Node	Multiple entries identified by unique key	`interface* [name]`
Leaf	Single data value node under a container or list	`name`, `type`, `enabled`, etc.

CLI and navigation notes:

- Use the **show yang operational** command to explore down to the leaf level in a data model, similar to navigating hierarchical data.
- Data model structure guides how CLI outputs and configurations map to YANG-defined nodes.

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

Figure 1: Mapping between CLI and Data model



Navigating YANG operational data models via CLI commands

YANG operational data models can be queried and explored using various CLI commands on Cisco network devices. These commands help you efficiently access operational state data, identify container and node structures, and retrieve detailed model attributes for troubleshooting and monitoring. Below are the main features, queries, outputs, and usage instructions for navigating YANG operational data.

Table 3: Key queries for YANG operational data

Operational Query	Description

Search specific top-level nodes	Search and produce the output of keywords from top-level nodes.
	`Router#show yang operational`
	`Router#show yang operational | include <component>`
	The following example shows the search result for interfaces:
	`Router#show yang operational | include interface`
	`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`

Operational Query	Description

All the instances of the container	Lists all the models at the root level container and its container name.
	`Router#show yang operational ?`
	You can also see the containers for a partially typed keyword. For example, keyword search for `mpls-` displays all the containers with `mpls` :
	`Router#show yang operational mpls-`
	`mpls-io-oper-mpls-ea mpls-io-oper-mpls-ma`
	`mpls-ldp-mldp-oper:mpls-mldp`
	`mpls-lsd-oper:mpls-lsd mpls-lsp-oper:mpls-lsd-nodes`
	`mpls-ldp-mldp-oper:mpls-mldp`
	`mpls-vpn-oper:l3vpn mpls-te-oper:mpls-tp`
	`mpls-te-oper:mpls-te`
	View the container data. The output of the command is in-line with the structure of the data model.
	`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`
	`}`
	`}`
	`}`

Operational Query	Description

All the nodes of the container	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>
	Output in JSON Format:
	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"
	}
	]
	}
	}
	}
	Output in XML Format:
	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>

Operational Query	Description

Navigate until the last leaf level	`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-enbled-interface-count`
	`vrf-count`
	`| Output Modifiers`
	`<cr>`
	View data specific to the leaf value. The `read only` (ro) leaves in a YANG model are considered as the state data (operational).
	`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": [`
	`}`
	`}`
	`}`

Model-Driven CLI to display running configuration in XML and JSON formats

A model-driven CLI display format is a configuration output method that

- presents device running configuration in structured XML and JSON formats,
- leverages native, OpenConfig, and unified YANG data models to organize configuration data, and
- enables granular filtering and flexible retrieval of configuration for specific components or hierarchies.

Table 4: Feature History Table

Feature Name	Release Information	Description
Model-driven CLI to Display Running Configuration in XML and JSON Formats	Release 25.1.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100])(select variants only*) *This feature is now supported on Cisco 8712-MOD-M routers.

Feature Name	Release Information	Description
Model-driven CLI to Display Running Configuration in XML and JSON Formats	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100])(select variants only*); Modular Systems (8800 [LC ASIC: P100])(select variants only*) *This feature is now supported on: • 8212-48FH-M • 8711-32FH-M • 88-LC1-12TH24FH-E • 88-LC1-36EH+A8:B12 • 88-LC1-52Y8H-EM
Model-driven CLI to Display Running Configuration in XML and JSON Formats	Release 7.3.2	This feature enables you to display the configuration data for Cisco IOS XR platforms in both JSON and XML formats. This feature introduces the **show run | [xml | json]** command.

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 for the show run command

The **show run | xml** command produces the router's running configuration in XML format. This enables automation and integration with software that can parse device settings through a standard XML schema.

```
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 for show run command

he `show run | json` command on Cisco IOS XR displays the device's running configuration in JSON format, making it easier for automation tools, scripts, or APIs to parse and manipulate the configuration data programmatically.

A typical output includes:

- Interface configuration details, such as interface name, state, and protocol.
- Netconf/YANG agent configuration, such as SSH status.

```
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 for show run command

The output of the **show run router bgp <as-number> neighbor <neighbor-address> | json unified** command provides a structured, highly detailed view of BGP configuration for a specified neighbor in JSON format. This reference explains how to interpret each component of the output and highlights key attributes.

```
Router#show 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 for show run command

The following sample output demonstrates the unified model XML structure produced by the **show run router bgp 12 | xml unified** command:

```
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>
```

This output helps you identify the XML element hierarchy and data structure when configuring BGP settings using the unified model on Cisco IOS XR devices.