Catalyst 9800 Programmability and Telemetry Deployment Guide

November 2021

Contents

Feature overview

The world of programmability has been evolving for years, and with the latest Cisco® IOS XE releases, we've included new Yet Another Next Generation (YANG) models to bring additional automation to wireless technology. With the use of APIs, interacting with devices and retrieving data got much easier. Back in the day, we used many commands sent from a Command Line Interfaces (CLIs) to communicate with the software. In addition, the Simple Network Management Protocol (SNMP) is frequently used for network management. Fast forward to today, we use the new way to interact with the software commonly called an API. Even though CLIs and SNMP are widely used, they are not as efficient and scalable as APIs.

This document will dive into the different programmable interfaces used to communicate with the various Cisco devices and specifically target the Catalyst® 9800 Wireless LAN Controller. We will discuss the pros and cons of using Network Configuration Protocol (NETCONF), Representational State Transfer Configuration (RESTCONF), and the gRPC Network Management Interface/Google Remote Procedure Call (gNMI/gRPC) protocols, and the main differences between them.

The Catalyst 9800 IOS XE–based Wireless LAN Controller (WLC) has several options for programmatic configuration. Traditional methods for configuring the WLC include the CLI and WebUI, but that has now been expanded to include the programmatic interfaces. These programmatic interfaces include NETCONF, RESTCONF, and the gNMI/gRPC protocols. YANG data models define what data is accessible over the programmatic interfaces, and they come in several varieties. Cisco IOS XE features are defined within the Native data models, while standard and vendor agnostic features are defined within the open data models that are mapped to Native data models. Either model can be used for many tasks; however, features specific to IOS XE are available only in the Native models.

Models created by Cisco are referred to as Native data models since they are specifically created for devices and software with which they are associated. The Native data models provide most comprehensive and operational coverage for device functionality.

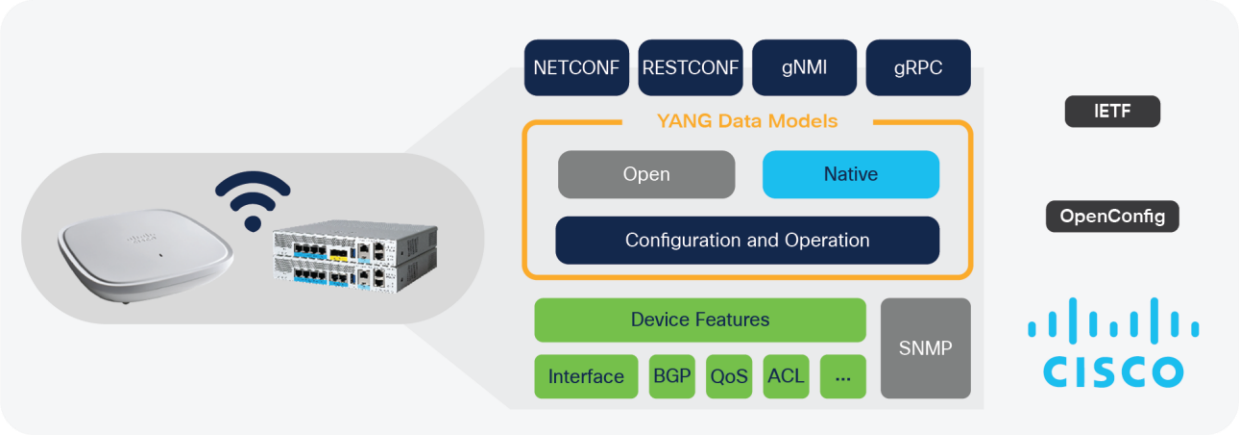


Figure 1.
Wireless model-driven telemetry interfaces

Telemetry history

SNMP and CLI have been used as traditional network management tools for years. Since the technology is widely popular in network management for network monitoring, network engineers still actively use it. All the SNMP messages are transported via User Datagram Protocol (UDP) and always require active polling by the collector, which is not fully reliable. The read-write mode can make a network vulnerable to attacks and can be used to access it. Other limitations include the security concerns of SNMP, lack of information about the source IP address, what type of traffic is sent, and the information about the destination IP address. In other words, the data is mainly unstructured, and the format frequently changes between each software release. Nevertheless, more recent versions of SNMP bring improvements in security, performance, and flexibility.

In contrast to SNMP, NetFlow was designed for network monitoring. It brings more visibility into the network and explicitly gives information about the source IP address, application protocol, and destination IP address, which SNMP lacks. NetFlow works in the same way as SNMP, it sends records from a cache to a collector, and all the records are being pushed to the collector without being requested each time. However, NetFlow is not used for collecting information about bandwidth, CPU utilization, memory, and the temperature of the device apart from the SNMP functionalities.

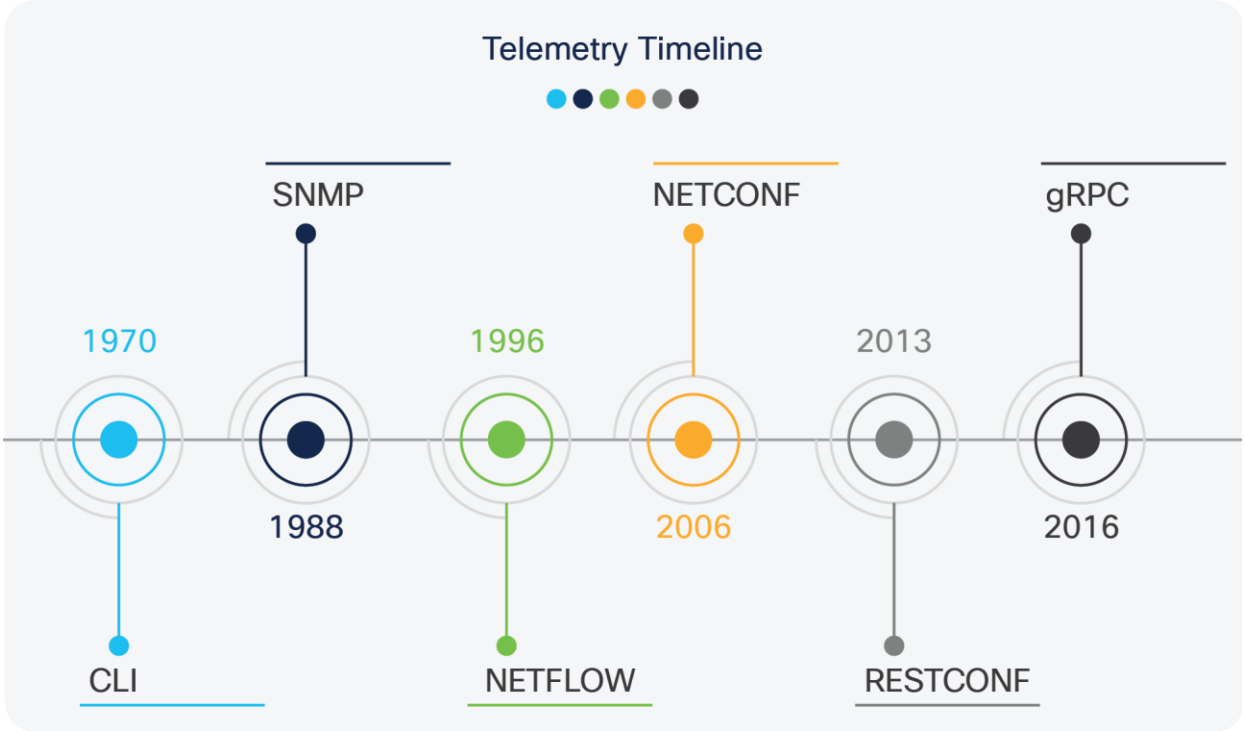


Figure 2.
Telemetry standards timeline

For the past couple of years, we have adopted new technologies that helped us solve predecessors' flaws in the modern world, but not all. Cisco IOS XE supports the YANG data modeling language, which can be used with the NETCONF to deliver the desired programmable and automated network operations. NETCONF is an XML-based protocol used to install, manipulate, and delete the configuration of network devices via Secure Shell (SSH) as the transport layer. It is based on an RPC (Remote Procedure Call) mechanism to provide communication between a client and a server. In our case, the server is the network device—Cisco Catalyst 9800 Wireless LAN Controller—and the controller is the client.

The Web is progressing at an exponential speed, and that's when we started seeing a quick adoption of RESTCONF in the Web-based monitoring stack. RESTCONF uses HTTP methods to implement similar NETCONF operations for accessing data defined in YANG. In comparison to NETCONF, RESTCONF supports both XML and JavaScript Object Notation (JSON) encodings. But it's important to clarify that RETCONF is not a NETCONF replacement and has never been intended to be one. From a capabilities standpoint, RESTCONF has its limitations like any other network automation tool. It lacks any way of validation and also lacks the "lock" concept found in NETCONF.

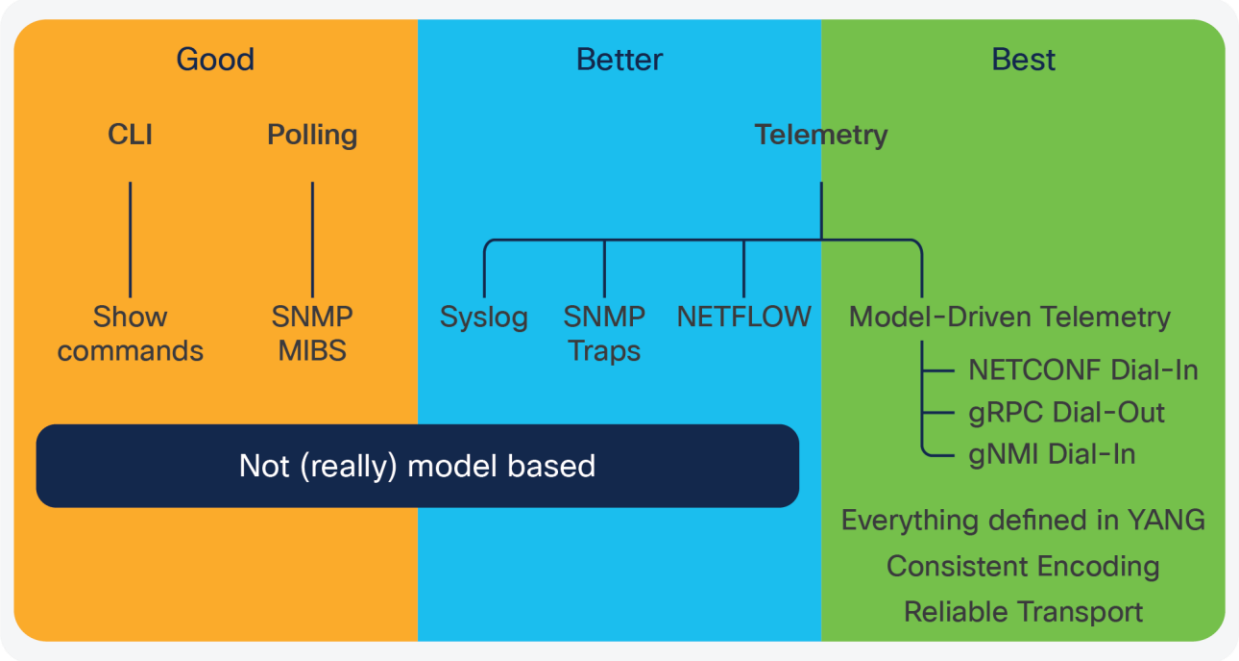


Figure 3.
Comparison chart

In addition, we have gRPC or Google Remote Procedure Call. It is a modern open-source RPC using HTTP for APIs. Model-driven telemetry with gRPC addresses many of the shortfalls of the legacy monitoring capabilities and provides an additional interface that telemetry is now available to be published from. gRPC is a YANG model using JSON and Protobuf encodings. Unlike NETCONF telemetry interface, which is "dial-in" and session-based, the gRPC interface is "dial out" and is based on configuration within the Catalyst 9800 device. gRPC is push-based, meaning that once it is configured, it will send the requested telemetry data regularly to the provided recipient(s) without them needing to request the data. Now you decide what data is needed, how often, and where to send it to. Once the configuration is in place, the Cisco IOS XE device happily publishes the telemetry data to the third-party collectors, your monitoring tools, extensive data search and visualization engines such as Splunk and Elastic, or even a simple text file.

Finally, while the CLI and SNMP aren't going away anytime soon, automation is a big part of where networks are headed. Protocols like YANG, NETCONF, RESTCONF, and gRPC were designed with this in mind. That's why Cisco uses them within their platforms.

Cisco device compatibility

Table 1. Catalyst 9800 WLC IOS XE operational consistency

	CAT9100-EWC	CAT9800-CL	CAT9800-L	CAT9800-40/80
Provisioning Automation				
ZTP	No	No	17.7.1	16.12+
Model-Driven Configuration Management				
NETCONF	16.12+	16.10+	16.12+	16.10+
RESTCONF	16.12+	16.11+	16.12+	16.11+
gNMI	No	17.6+	17.6+	17.6+
Model-Driven Telemetry				
NETCONF Dial-In	16.12+	No	No	No
gRPC Dial-Out	17.6	17.6	17.6	17.6
gNMI Dial-In	17.6	17.6	17.6	17.6
gNMI explicit wildcard	No	No	No	No
Software Image Management				
GuestShell (On box Python)	No	No	17.7.1	16.11+
gRPC Network Operations Interface (gNOI)				
cert.proto	No	17.6	17.6	17.6
os.proto	No	No	No	No
reset.proto	No	No	No	No

Configuration overview

Regardless of the interfaces used, the following configuration is required on the Catalyst 9800:

```
configure terminal
aaa new-model
aaa authentication login default local
aaa authorization exec default local
exit
write memory
```

Authentication requirements

A user account is required to access the programmatic interfaces. This could come in the form of an existing or preconfigured “username” account because a dedicated NETCONF or RESTCONF account is not required. Alternately, an existing or pre-configure “admin” account can be used. Authentication via Terminal Access Controller Access Control System Plus (TACACS) or RADIUS is also supported if the user is granted full or privilege Level 15 rights upon login. To create an additional user account with username netconf or restconf and password netconf or restconf, use the following commands when using local authentication:

```
username netconf privilege 15 password 0 netconf
username restconf privilege 15 password 0 restconf
```

NETCONF Access Control Module (NACM) and Model-Based AAA

The programmatic interfaces support NETCONF Access Control Model (NACM), which is a form of Role-Based Access Control (RBAC) that is defined in RFC6536. This is commonly referred to as Model-Based AAA. Use this feature to create rules for users that are logging in over the programmatic interfaces so that access to certain models or functions can be permitted or denied as needed. More details can be found in the “Model-Based AAA” chapter of the IOS XE Programmability User Guide, which is linked in the additional resources section.

API - Role-Based Access Control

Prior to the 17.5 release, a user with privilege Level 15 was required for any NETCONF operations. In the 17.5+ releases, support for a lower privileged and read-only user has been introduced. Whenever a lower privileged user attempts to access information such as username or password, that sensitive data gets masked and not visible to the user.

To enable the API RBAC, enter the following commands:

```
Device#configure terminal
Device(config)#username <USERNAME> privilege 1 password <PASSWORD>
Device(config)#end
Device#request platform software yang-management nacm populate-read-rules privilege 1
```

To get configs as priv1 user when the NACM rules are populated, use the following command:

```
netconf-console --host <IP_ADDRESS_OF_THE_DEVICE> --port 830 -u <USERNAME> -p <PASSWORD> --
get-config
```

```
Desktop % netconf-console --host 10.0.0.237 --port 830 -u priv1 -p priv1 --get-config
<?xml version='1.0' encoding='UTF-8'?>
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <app-hosting-cfg-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-app-hosting-cfg">
    <apps>
      <app>
        <application-name>guestshell</application-name>
        <application-network-resource>
          <management-interface-name>1</management-interface-name>
        </application-network-resource>
      </app>
      <app>
        <application-name>GuestShellApp</application-name>
        <application-network-resource>
          <management-interface-name>0</management-interface-name>
        </application-network-resource>
      </app>
    </apps>
  </app-hosting-cfg-data>
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
    <version>17.5</version>
```

When specifying the NETCONF read-only privilege level:

- Allowed RPCs are set - **`get, get-config, get-schema`**.
- Sensitive information is masked - Native/enable, native/aaa, native/username.
- Read-only access to all models is provided.

YANG data models

YANG is a data modelling language for NETCONF, RESTCONF, and gNMI. YANG models within the IOS XE device have been defined to describe how to structure the data to send or receive. The YANG standard was defined in RFC6020. There are two main types of YANG models in use: Native and Open. The models are further categorized as either Configuration or Operational models. The configuration models can be used for programmatic configuration, while the operational models can be used with telemetry to show real-time operational data.

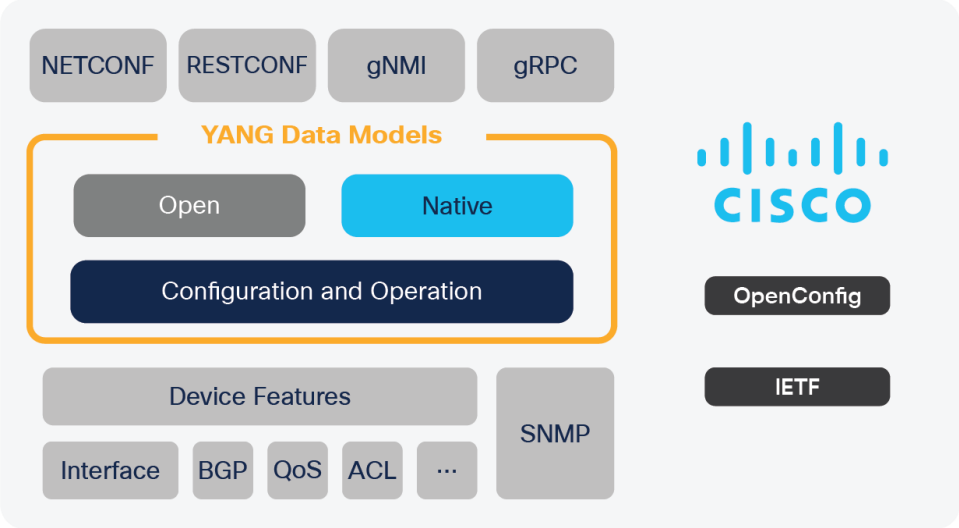


Figure 4.
Model-driven telemetry interfaces

Native models

The Native models for wireless can be grouped into two main categories: config and operational. The config modules contain configuration information for the related features, while the operational models provide run time and operational data about the feature. All of the XPath expressions listed below are a part of the openconfig-access-points YANG model, except the last one, which is a part of the openconfig-ap-manager YANG model. For the telemetry operation to work correctly, ensure that configurations are done based on the OpenConfig model.

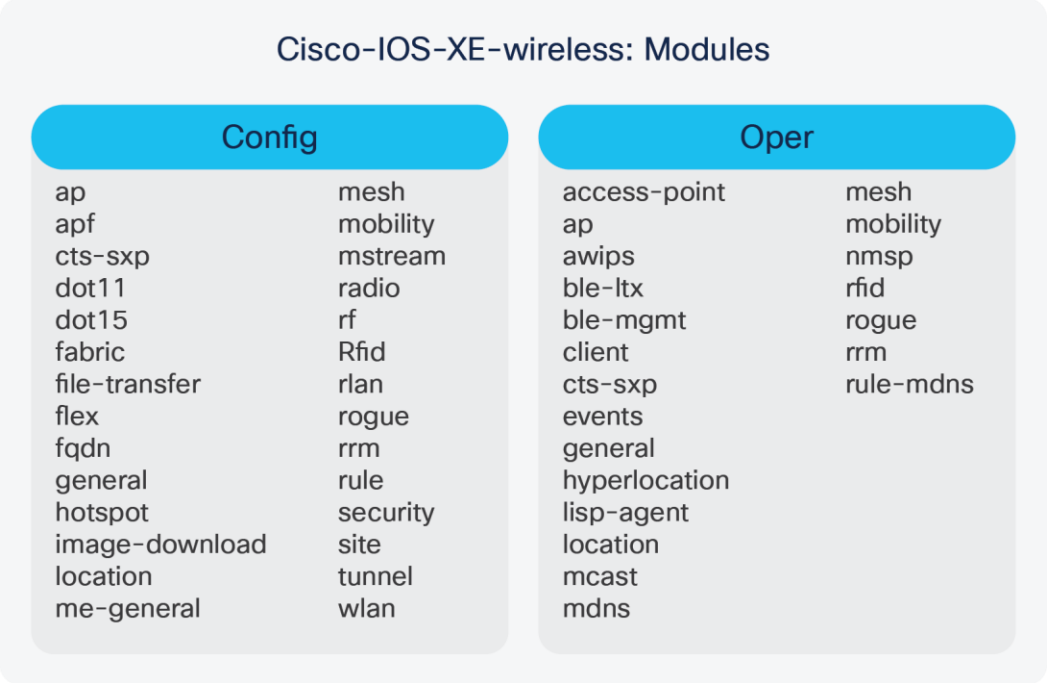


Figure 5.
Wireless Native config and oper models

Native configuration models

Table 2. Native configuration models

Config Module	Data Model Details
ap	AP configuration: tags, policy tags, rf tags, site tags
apf	Global dot11 parameters: country, association request limit, blocklisting
cts-sxp	CTS SXP configuration connection information, SXP mode, password, profile name, hold times, retry periods
dot11	802.11 configuration: TX Powe, 802,11 n/802.11ac features, admin state data rates
dot15	802.15 global configurations: global 802.15 radio switch
fabric	Fabric configuration: list of trace export profiles, transfer parameters for profiles, config download
file-transfer	File transfer configuration: list of trace export profiles, transfer parameters for profiles, config download

Config Module	Data Model Details
flex	Flex configuration: Remote LAN (RLAN) configuration, VLAN ALCs
fqdn	Fully qualified domain name (FQDN) configurations: url-filter, domain names, filter action
general	Misc WLC configurations: AP join configs, wireless management interfaces, multicast, WSA config
hotspot	Configuration for the Wi-Fi Alliance Hotspot 2.0 (Passpoint) capabilities of the wireless controller, including configuration for 802.11u
image-download	Model for managing AP and Mobility Express Wireless Controller (MEWLC) image download configurations
location	Location configurations: Network Mobility Services Protocol (NMSP), CMX cloud, server URL, timeouts, measurement intervals
me-general	Model for managing Embedded Wireless Controller miscellaneous configuration
mesh	Wireless Mesh configurations: mesh profile, psk, aaa method, threshold values
mobility	Mobility configurations: mode, congif, peer config, group config, guest mode
mstream	Multicast configurations: group, url, group name, stream groups, priority
radio	Radio configurations: ap-specific configs
rf	RF configurations: allowed channels, rf-profile, data rates config, Dynamic Channel Assignment (DCA)
rfid	Radio Frequency Identification (RDIF) configurations: timeouts, Received Signal Strength Indicator (RSSI) expiry, notify thresholds
rogue	Rogue configurations: rogue global parameters, SSID rules, clients, ignore list
rrm	Radio Resource Management (RRM) configurations: intervals, grouping algorithm, static members
rule	Model for rule configuration for wireless application based on regular expression match
security	AP security configurations: AP Local Session Controller (LSC) config, CA trustpoint, key size
site	Site configurations: site tag, rlan ports, master AP, join profile, BLE, Access Point Packet Capture (AP PCAP), AP config profiles
tunnel	Wireless tunnel configuration: tunnel profiles, Ethernet over Soft Generic Routing Encapsulation (EoGRE) tunnel domain, AAA tunnel proxy for EoGRE
wlan	WLAN configurations: aaa, dhep, umbrella, 802.11v, flex profiles, WLAN SSID, 802.lx security

Native operational models

Table 3. Native operational models

Config Module	Data Model Details
access-point	Join failure, discovery, certificates, country
ap	AP configuration: tags, policy tags, rf tags, site tags
awips	This module contains a collection of YANG definitions for SWIPS (Adaptive Wireless Intrusion Prevention Service) operational data.
ble-ltx	This module contains a collection of YANG definitions for wireless Bluetooth Low Energy (BLE) LTX operational data.
ble-mgmt	This module contains a collection of YANG definitions for wireless Bluetooth Low Energy (BLE) management operational data.
client	Session details, eapol, wlan id, delete reasons, association details
cts-sxp	This module contains a collection of YANG definitions for AP Scalable Group Tag (SGT) Exchange Protocol (SXP) operational data.
events	This module contains a collection of YANG definitions for wireless operations events generated from the Cisco Wireless controller. These models may produce high volume of data.
general	This module contains a collection of YANG definitions for general operational data.
hyperlocation	This module contains a collection of YANG definitions for wireless Hyperlocation operational data.
lisp-agent	Wireless Fabric Control Plane oper data
location	Model for accessing location data for clients. This module produces a large amount of data. The update rate will be high for this data.
mcast	Multicast
mdns	This module contains a collection of YANG definitions for Multicast DNS (MDNS) gateways operational data. This operational data consists of consolidated MDNS packet statistics and per WLAN MDNS packet statistics.
mesh	Mesh
mobility	Mobility
nmsp	NMSP location
rfid	Radio Frequency Identification (RFID)
rogue	Rogues
rrm	Radio Re
rule-mdns	This module contains a collection of YANG definitions for rule MDNS operational data

Open models

In addition to the Cisco Native configuration and operational models, there are several additional YANG models that are supported on the device. The capabilities exchange via NETCONF, RESTCONF and gNMI YANG library. There are models for SNMP MIBs, IETF, and OpenConfig. These models can be used in the same way as the Native models; however, they offer a limited or subset of the capabilities available on the device.

NETCONF telemetry and programmatic interface

NETCONF is a protocol defined by the Internet Engineering Task Force (IETF) to “install, manipulate, and delete the configuration of network devices.” NETCONF operations are realized on top of a Remote Procedure Call (RPC) layer using an XML encoding and provide a basic set of operations to edit and query configuration and operational state on a network device.

NETCONF configuration

To enable the NETCONF interface, enter the following commands:

```
Device# configure terminal
Device(config)# netconf-yang
Device(config)# exit
```

NETCONF-YANG SSH server support

NETCONF-YANG uses the IOS Secure Shell and can be configured to use Rivest, Shamir, and Adelman (RSA) public keys to authenticate users as an alternative to password-based authentication.

For public-key authentication to work on NETCONF-YANG, the IOS SSH server must be configured. To authenticate users to the SSH server, use one of the RSA keys configured, by using the **`ip ssh pubkey-chain`** and user commands.

NACM is a group-based access control mechanism. When users are authenticated, they are automatically placed in an NACM privilege group based on their configured privilege level. Users can also be manually placed in other user-defined groups. The default privilege level is 1. There are 16 privilege levels, PRIV00 to PRIV15.

If a user authenticates via the public-key; but does not have a corresponding Authentication, Authorization, and Accounting (AAA) configuration, this user is rejected. If a user authenticates via a public key; but the AAA configuration for NETCONF is using a AAA source other than the local, this user is also rejected. Local and Terminal Access Controller Access-Control System Plus (TACACS+) AAA authorization are supported.

An example of how to allow local login in the console line, and NETCONF to authenticate and get authorization from TACACS can be found here: https://github.com/jeremycohoe/netconf-tacacs-aaa.

YANG models

The NETCONF capabilities exchange can be retrieved by connecting to the device on the default port TCP 830, using ssh. The capabilities exchange lists all available YANG data models supported by the device. Using tools like YANG Suite, which is detailed in further sections, these YANG modules can be downloaded from the device and analyzed further.

```
                    ~ % ssh admin@10.0.0.237 -p 830
admin@10.0.0.237's password:
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
<capability>urn:ietf:params:netconf:base:1.0</capability>
<capability>urn:ietf:params:netconf:base:1.1</capability>
<capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
<capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
<capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
<capability>urn:ietf:params:netconf:capability:notification:1.0</capability>
<capability>urn:ietf:params:netconf:capability:interleave:1.0</capability>
```

Verifying NETCONF status

To verify the NETCONF interface is operational, run the command:

```
Device#show platform software yang-management process
```

Ensure the NETCONF SSHD “ncsshd” process is running

```
C9800-WLC-L-C#show platform software yang-management process
confd            : Running
nesd             : Running
syncfd           : Running
ncsshd           : Running
dmiauthd         : Running
nginx            : Running
ndbmand          : Running
pubd             : Running
gnmib            : Not Running
```

NETCONF XML RPC payload examples

The XML RPC payloads below can be generated, modified, and sent from within the Cisco YANG Suite, or any other tool that can send XML payloads over the NETCONF interface, such as a Python script. Example WLC creation, verification, and removal XML payloads are listed below. These can be used to create, verify, and remove a WLAN over the NETCONF interface quickly and easily.

WLAN creation

The XML RPC below can be sent over the NETCONF interface to create a WLAN with the defined parameters. In this case, the WLAN ID is 4 and the SSID is “open.”

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <wlan-cfg-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-wireless-wlan-cfg">
        <wlan-cfg-entries>
          <wlan-cfg-entry>
            <profile-name>Open</profile-name>
            <wlan-id>4</wlan-id>
            <security-wpa>true</security-wpa>
            <wpa2-enabled>true</wpa2-enabled>
            <wpa2-aes>true</wpa2-aes>
            <auth-key-mgmt-dot1x>true</auth-key-mgmt-dot1x>
            <apf-vap-id-data>
              <broadcast-ssid>true</broadcast-ssid>
              <ssid>open</ssid>
            </apf-vap-id-data>
          </wlan-cfg-entry>
        </wlan-cfg-entries>
        <wlan-policies>
          <wlan-policy>
            <policy-profile-name>open</policy-profile-name>
            <status>true</status>
          </wlan-policy>
        </wlan-policies>
      </wlan-cfg-data>
    </config>
  </edit-config>
</rpc>
```

WLAN verification

The XML RPC below can be sent over the NETCONF interface to verify a WLAN with ID 4.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <get>
    <source>
      <running/>
    </source>
    <filter>
      <wlan-cfg-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-wireless-wlan-cfg">
        <wlan-cfg-entries>
          <wlan-cfg-entry>
            <profile-name>Open</profile-name>
            <wlan-id>4</wlan-id>
          </wlan-cfg-entry>
        </wlan-cfg-entries>
      </wlan-cfg-data>
    </filter>
  </get>
</rpc>
```

WLAN removal

The XML RPC below can be sent over the NETCONF interface to remove a WLAN with ID 4.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <wlan-cfg-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-wireless-wlan-cfg">
        <wlan-cfg-entries xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
nc:operation="delete">
          <wlan-cfg-entry>
            <profile-name>Test-WLAN</profile-name>
            <wlan-id>4</wlan-id>
          </wlan-cfg-entry>
        </wlan-cfg-entries>
      </wlan-cfg-data>
    </config>
  </edit-config>
</rpc>
```

NETCONF Dial-In Model-Driven Telemetry

Dynamic Telemetry Subscription with Python NCC

Dial-in telemetry subscriptions can be easily created by using the NCC tools available from CiscoDevNet at https://github.com/CiscoDevNet/ncc. The repository can be simply cloned from Github with the **git clone** command. An additional requirement is to use a patched ncclient tool that works with the ncc-establish-subscription.py tool. This can be installed by following the directions on the ncc Github page or by executing the two commands below:

```
$ git clone https://github.com/CiscoDevNet/ncc
$ sudo pip install --upgrade git+https://github.com/CiscoDevNet/ncclient.git
```

Once downloaded, a subscription can be established by running the following command:

```
$ python3 ncc-establish-subscription.py  --host <HOST_IP_ADDRESS> --port 830 -x "/wireless-client-oper:client-oper-data/dot1-oper-data" --period 1000 -u <USERNAME> -p <PASSWORD>
```

```
ncc % python3 ncc-establish-subscription.py --host 10.0.0.237 --port 830 -x "/wireless-client-oper:client-oper-data/dot11-oper-data" --period 1000 -u admin -p cisco
Subscription Result : notif-bis:ok
Subscription Id     : 2147483648
-->>
(Default Callback)
Event time      : 2021-07-21 18:20:03.440000+00:00
Subscription Id : 2147483648
Type            : 1
Data            :
<datastore-contents-xml xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push"/>
```

In the example above, a telemetry subscription has been created using the Cisco-IOS-XE-wireless-client-oper YANG model, which has a prefix of “wireless-client-oper” and an xpath of “/client-oper-data/dot11-oper-data.” YANG Suite is used to determine the correct XPath Filter for this YANG model as seen in the screenshot below:

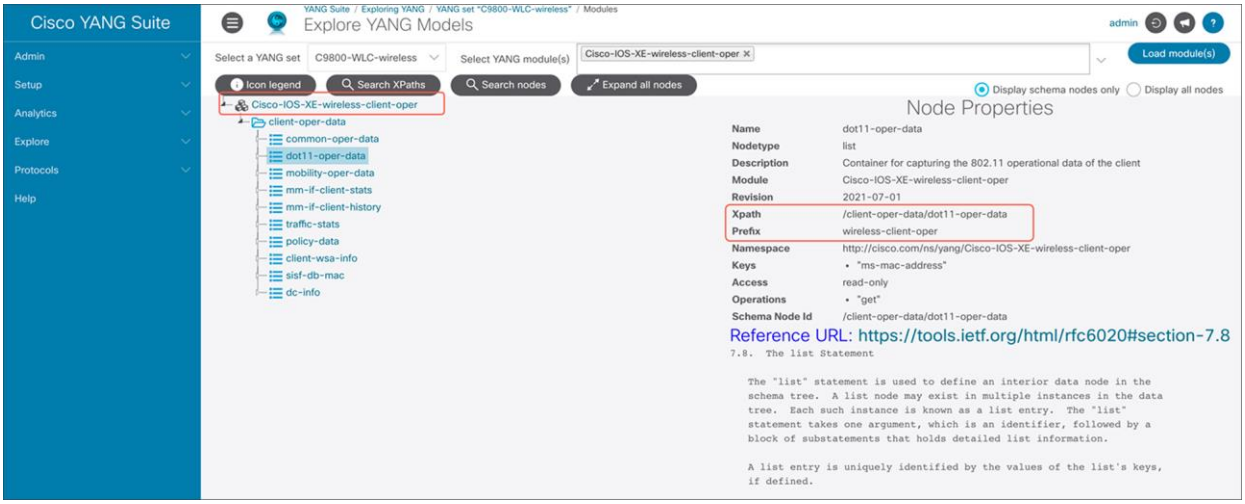


Figure 6.
Cisco YANG Suite, Explore YANG Models dashboard

RESTCONF programmatic interface

RESTCONF stands for the HTTP-based Representational State Configuration Protocol (RFC 8040). It is a stateless protocol that uses the secure HTTP method. RESTCONF uses structured XML or JSON and YANG data models to provide a REST-like API that enables programmatic access to the network device. RESTCONF API uses HTTPs method and commands like PUT and GET to send information to and from the Cisco devices. The Catalyst 9800 IOS XE implementation supports the following RESTCONF operations: `GET, PATCH, PUT, POST, DELETE, HEAD`.

RESTCONF configuration

To enable the RESTCONF interface on the device, enter the following commands:

```
configure terminal
ip http secure-server
restconf
exit
```

RESTCONF Verification

Verify that RESTCONF is running by sending the GET request to the device:

```
curl -k https://<IP ADDRESS>/restconf/ -u "username:password"
```

If everything was configured properly, you should get the following response:

```
<restconf xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <data/>
  <operations/>
  <yang-library-version>2016-06-21</yang-library-version>
</restconf>
```

RESTCONF YANG Models

The list of supported YANG data models can be retrieved from the RESTCONF interface by sending GET request to the URI `restconf/ietf-yang-library:modules-state`.

For example:

```
curl -k -u username:password https://<IP_ADDRESS>/restconf/data/ietf-yang-library:modules-state
```

```
~ % curl -k -u admin:cisco https://10.0.0.237/restconf/data/ietf-yang-library:modules-state

<modules-state xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library"  xmlns:yanglib="urn:ietf:params:xml:ns:yang:ietf-yang-library">
  <module-set-id>0a78e0ef6d7fda9b231d8915b710ab98</module-set-id>
  <module>
    <name>ATM-FORUM-TC-MIB</name>
    <revision></revision>
    <schema>https://10.0.0.237:443/restconf/tailf/modules/ATM-FORUM-TC-MIB</schema>
    <namespace>urn:ietf:params:xml:ns:yang:smiv2:ATM-FORUM-TC-MIB</namespace>
    <conformance-type>import</conformance-type>
  </module>
```

```
<module>
  <name>Cisco-IOS-XE-wireless-client-oper</name>
  <revision>2021-03-01</revision>
  <schema>https://10.0.0.237:443/restconf/tailf/modules/Cisco-IOS-XE-wireless-client-oper/2021-03-01</schema>
  <namespace>http://cisco.com/ns/yang/Cisco-IOS-XE-wireless-client-oper</namespace>
  <conformance-type>implement</conformance-type>
</module>
<module>
  <name>Cisco-IOS-XE-wireless-client-types</name>
  <revision>2021-03-01</revision>
  <schema>https://10.0.0.237:443/restconf/tailf/modules/Cisco-IOS-XE-wireless-client-types/2021-03-01</schema>
  <namespace>http://cisco.com/ns/yang/Cisco-IOS-XE-wireless-client-types</namespace>
  <conformance-type>implement</conformance-type>
</module>
```

A specific YANG module can be downloaded by sending another GET request to the URI containing the model. For example, to download the Cisco-IOS-XE-wireless-client-oper model send, the GET request to /restconf/tailf/modules/Cisco-IOS-XE-wireless-client-oper/2021-03-01 as seen in the screenshot below:

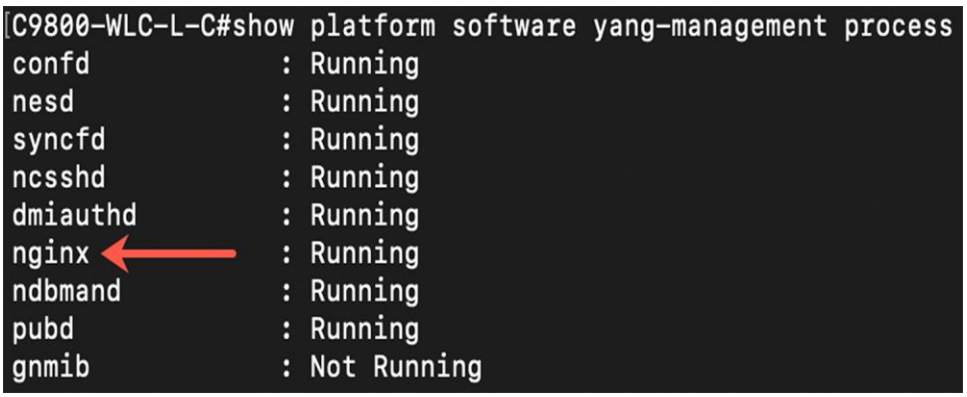
```
% curl -k -u admin:cisco https://10.0.0.237/restconf/tailf/modules/Cisco-IOS-XE-wireless-client-oper/2021-03-01
module Cisco-IOS-XE-wireless-client-oper {
  yang-version 1;
  namespace "http://cisco.com/ns/yang/Cisco-IOS-XE-wireless-client-oper";
  prefix wireless-client-oper;

  import Cisco-IOS-XE-wireless-client-types {
    prefix wireless-client-types;
  }
  import Cisco-IOS-XE-wireless-enum-types {
    prefix wireless-enum-types;
  }
  import Cisco-IOS-XE-wireless-mobility-types {
    prefix wireless-mobility-types;
  }
  import Cisco-IOS-XE-wireless-types {
    prefix wireless-types;
  }
  import ietf-inet-types {
    prefix inet;
  }
  import ietf-yang-types {
    prefix yang;
  }
  import cisco-semver {
    prefix cisco-semver;
  }
```

Verifying RESTCONF status

To verify the RESTCONF interface is operational, run the command:

```
show platform software yang-management process
```

Ensure the “nginx” process is running:



RESTCONF Debugging

With the rise of programmable APIs for network devices, customers are looking to leverage NETCONF and RESTCONF APIs in their automation workflow for uses like deploying, provisioning, and monitoring. Thus, customers are now looking to include debugging uses as well, like generating log files for troubleshooting.

Starting from IOS XE 17.6, a support for YANG models exists for enabling debugging of specific wireless clients as well as generating trace logs for these clients. Therefore, customers can leverage NETCONF and RESTCONF in their debugging workflows.

Prerequisites

1. Enable NETCONF and RESTCONF.
2. To enable RESTCONF: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/174/b_174_programmability_cg/restconf_protocol.html
3. NETCONF is used by Cisco YANG Suit to download the YANG models. To enable NETCONF: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/174/b_174_programmability_cg/configuring_yang_datamodel.html

Enable wireless conditional debugging

Based on the either the MAC or IP address of a device we can generate debug messages using the **`Cisco-IOS-XE-wireless-actions-rpc.yang`** model. This is equivalent to entering the following CLI command:

```
debug platform condition feature wireless <mac/ip> <MAC/IP Address>
```

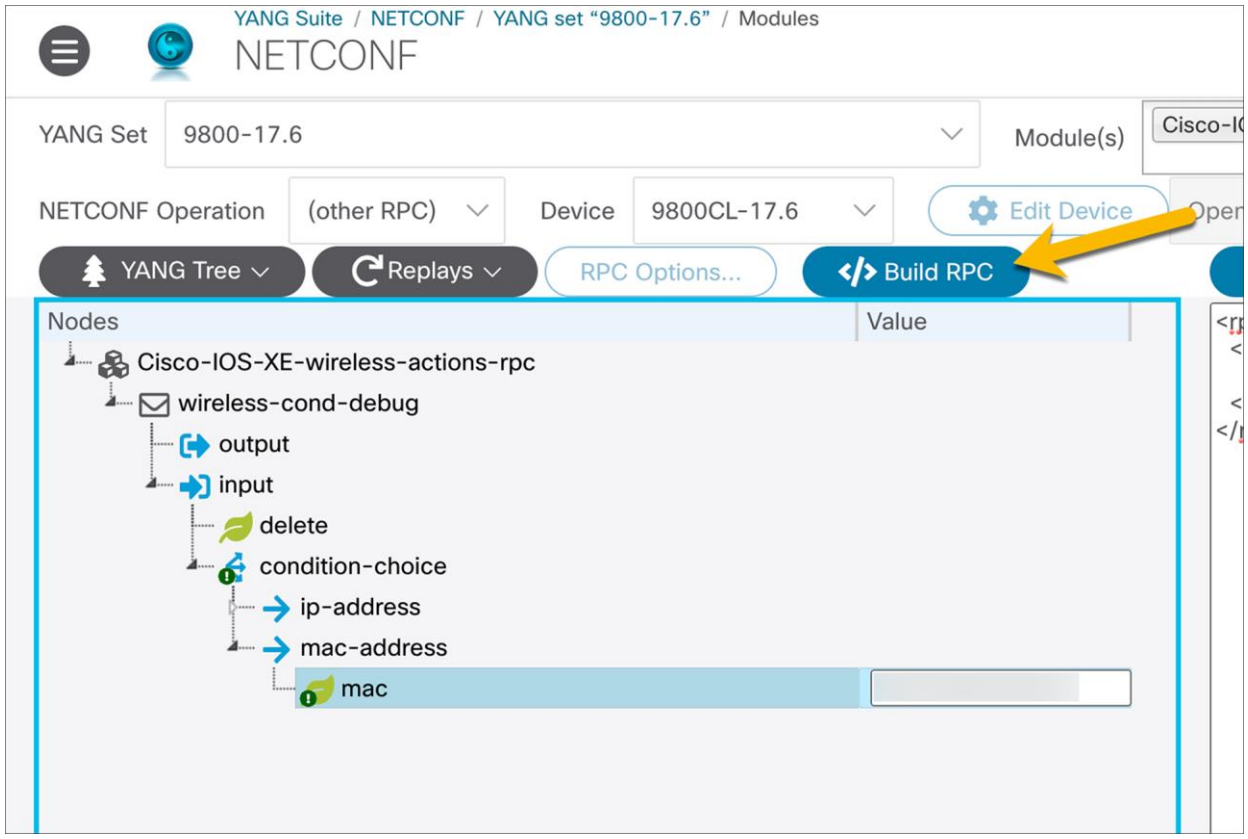


Figure 7.
How to build the NETCONF operation using the MAC address of an AP for the debugging

You can use YANG Suite to generate the RPC in XML form using NETCONF. To use the RPC for RESTCONF, the RPC will need to be converted from XML to JSON with any XML-specific information removed.

Generate trace archive

Using the Cisco-IOS-XE-trace-rpc.yang, we can generate a trace archive for the C9800 device. It is equivalent to the System Report which can be found in the C9800 WebUI under **Troubleshooting > Core > Dump** and **System Report**.

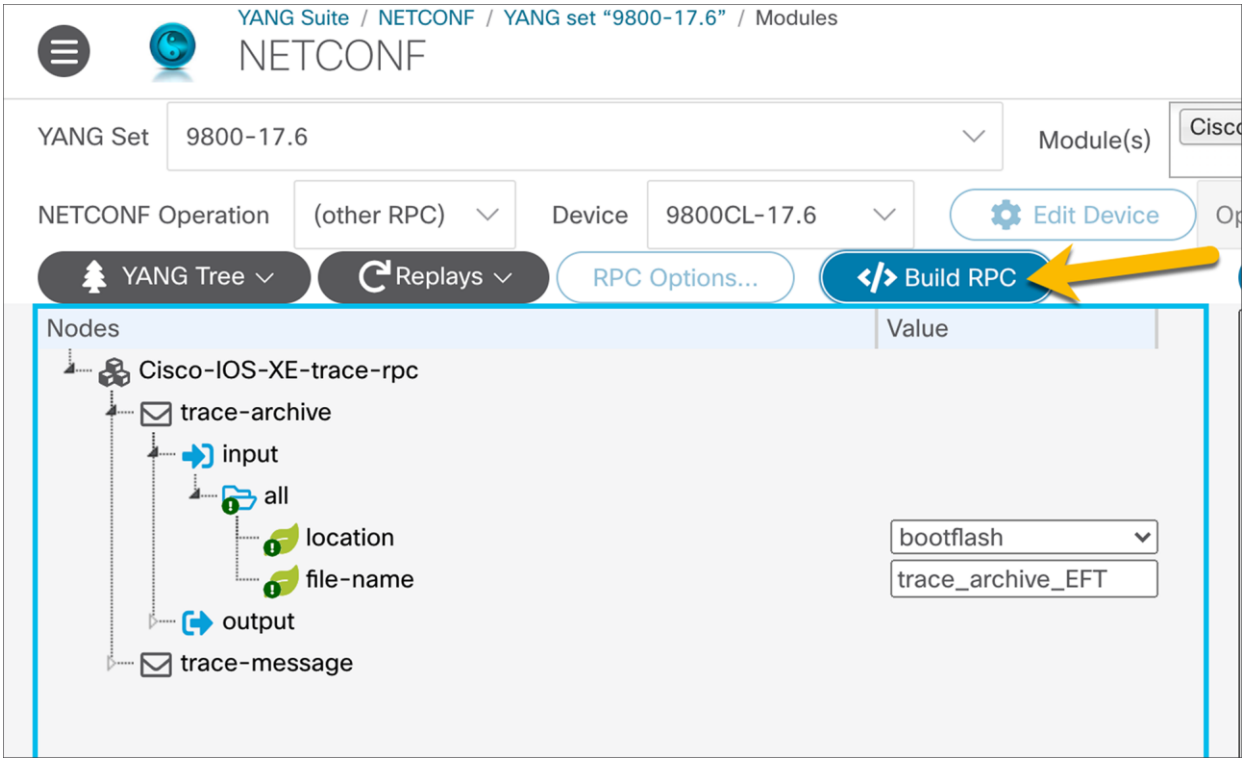


Figure 8.
How to build the NETCONF operation to generate the RPC in XML form for trace archive

In the NETCONF page of YANG Suite, build another RPC call to generate the RPC in XML form.

Note: The location where the trace archive will be saved depends on the model of C9800. This could be either **bootflash** or **harddisk.**

In order to use the RPC for RESTCONF, the RPC will need to be converted from XML to JSON with any XML specific information removed.

Generate trace message

Similar to a trace archive, the `Cisco-IOS-XE-trace-rpc.yang` module is used to generate trace debug messages based on the either the MAC or IP address of a device. It is equivalent to the Radioactive Trace which can be found in the C9800 WebUI under **Troubleshooting > Radioactive Trace**.

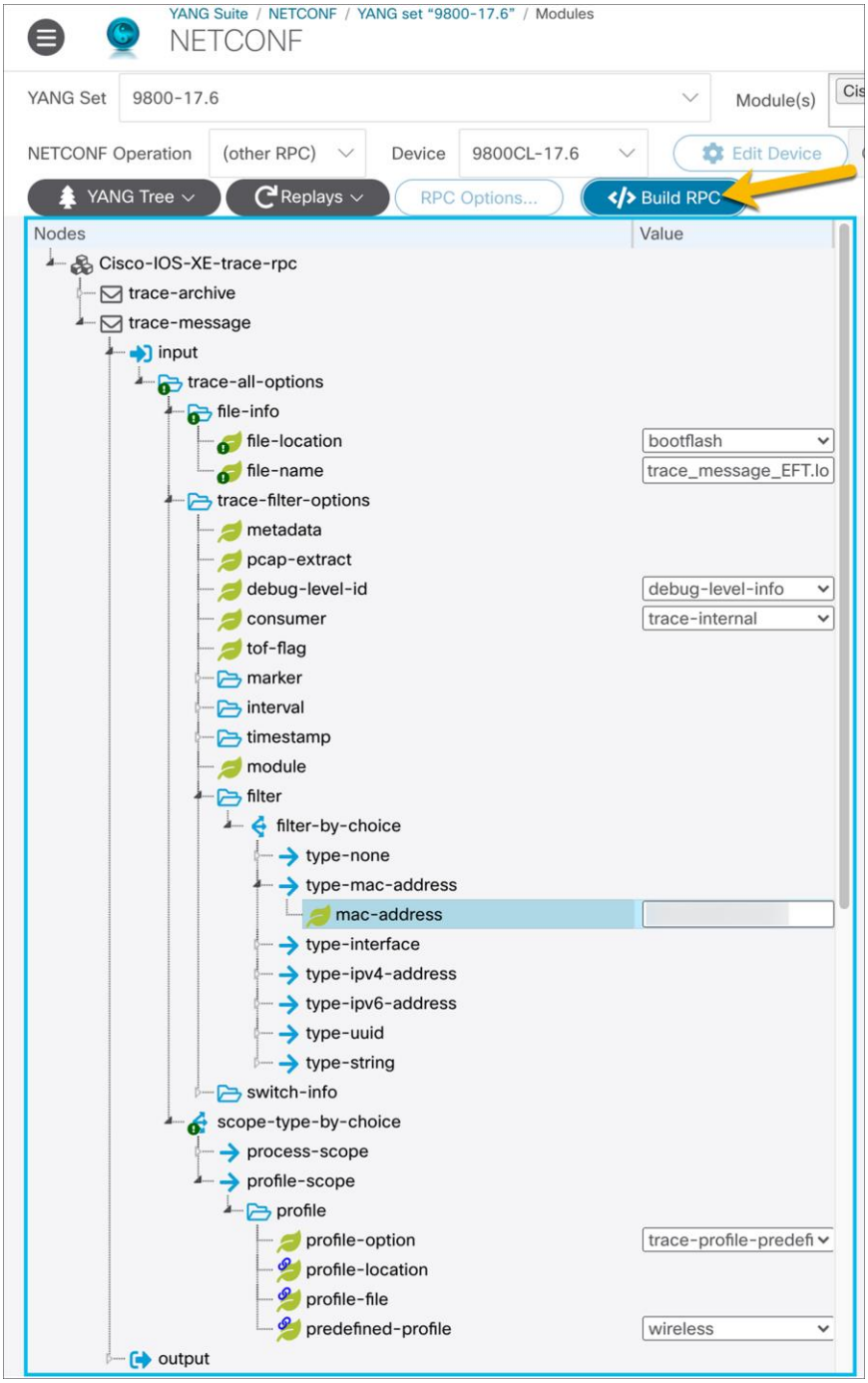


Figure 9.
How to build the NETCONF operation to generate the RPC in XML form for trace message

You can use YANG Suite to generate the RPC in XML form using NETCONF. In order to use the RPC for RESTCONF, the RPC will need to be converted from XML to JSON with any XML specific information removed.

The following trace-message leaves are not supported.

```
/trace-message/trace-all-options/trace-filter-options/metadata
/trace-message/trace-all-options/trace-filter-options/pcap-extract
/trace-message/trace-all-options/trace-filter-options/switch-info
/trace-message/trace-all-options/trace-filter-options/filter/integer-value
/trace-message/trace-all-options/trace-filter-options/switch-info
/trace-message/trace-all-options/trace-filter-options/filter/uuid-string
/trace-message/trace-all-options/trace-filter-options/filter/string-value
/trace-message/trace-all-options/profile/profile-location
/trace-message/trace-all-options/profile/profile-file
```

gRPC Dial-Out model driven telemetry

gRPC is a Remote Procedure Call dial-out model-driven telemetry interface. gRPC Dial-Out telemetry is an automated communications process by which measurements and other data are collected and transmitted to the remote receiving equipment for monitoring. Model-driven telemetry provides a mechanism to stream YANG-modeled data to a data collector over the network.

Dial-In Dynamic vs. Dial-out Configured MDT Subscriptions

With a dial-in or “dynamic” subscription, the subscriber must first establish session a connection to the device and then subscribe to the data models. The NETCONF session must remain established for telemetry data to remain streaming. If the session is disconnected, then the telemetry subscription must be manually re-established. With dial-out or “configured” subscriptions, once the configuration is setup by the user, the device will maintain the subscription configuration and send telemetry to the subscriber without needing an active session to the collector.

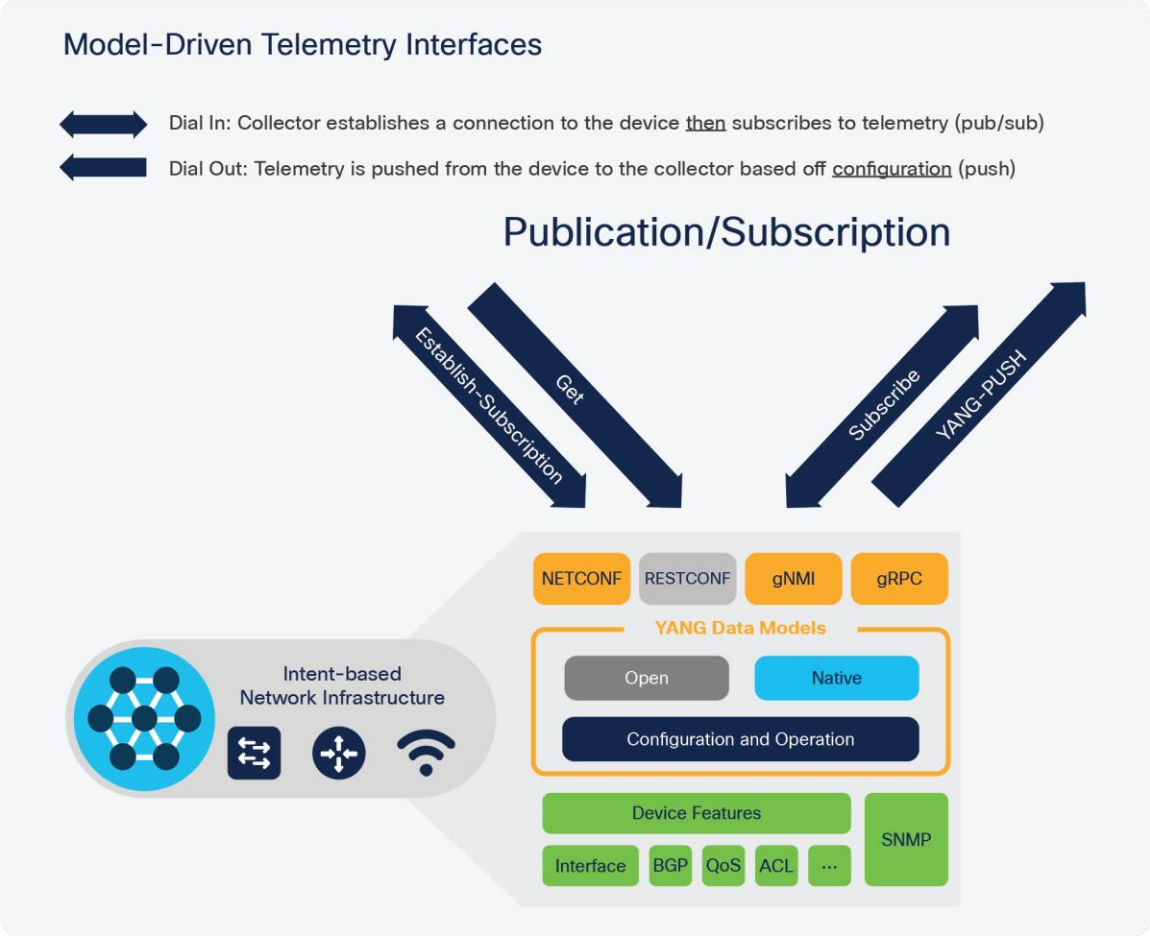


Figure 10.
Model-driven telemetry interfaces

gRPC Dial-Out Model-Driven Telemetry Configuration

gRPC Dial out subscriptions support encoding with key-value google protocol buffers (kv-gpb) over a TCP connection. The following configuration can be used to establish a gRPC dial-out with FQDN DNS support telemetry subscription. Configure domain lookup and name-server IP first and then add telemetry subscription with the DNS named receiver.

```
telemetry ietf subscription 101
 encoding encode-kvgpb
 filter xpath /wireless-client-oper:client-oper-data/dot11-oper-data
 source-address <IP_ADDRESS>
 stream yang-push
 update-policy periodic 2000
 receiver-type protocol
 receiver name yangsuite

telemetry receiver protocol yangsuite
 host name yangsuite-telemetry.cisco.com 57500
 protocol grpc-tcp
```

This configuration creates a new configuration subscription with an ID of 101. The encoding is set to kv-gpb, and the xpath filter defines the API to subscribe to, in this case dot11-oper-data. The xpath filter is defined within the YANG model, and YANG Suite is used to determine the exact xpath and prefix for this model. The source address and Virtual Routing and Forwarding (VRF) to use from the device is set, as well as the receiver IP, port, and protocol. The yang-push stream defines how often to publish data in centiseconds. In this case it is set to 1000, which means data will be published every 10 seconds. The following IOS XE CLI can be added to IOS XE 17.6 to enable the publication of Model-Driven Telemetry (MDT) on C9800. Make sure to change the source IP address and host name. Before enabling the publication of MDT, make sure to enable netconf-yang on the device.

Receiving gRPC Model-Driven Telemetry with Telegraf

The key-value encoded google protocol buffer (kv-gpb) telemetry data that is sent over the gRPC interface can be received with many tools and in many different configurations, depending on the business needs and use-cases. Telegraf is an open-source tool that can be used to receive the data and is available on Github at https://github.com/influxdata/telegraf. Telegraf works by acting as the gRPC server and receiver, where it processes the Google Protocol Buffers encoded data and sends the text data into the time series database InfluxDB. From there Grafana can be used to visualize the data. Telegraf and Grafana are highly configurable and can receive and visualize a variety of telemetry sources as well as output data to a variety of data source, including Kafka, InfluxDB, Elasticsearch, and Prometheus.

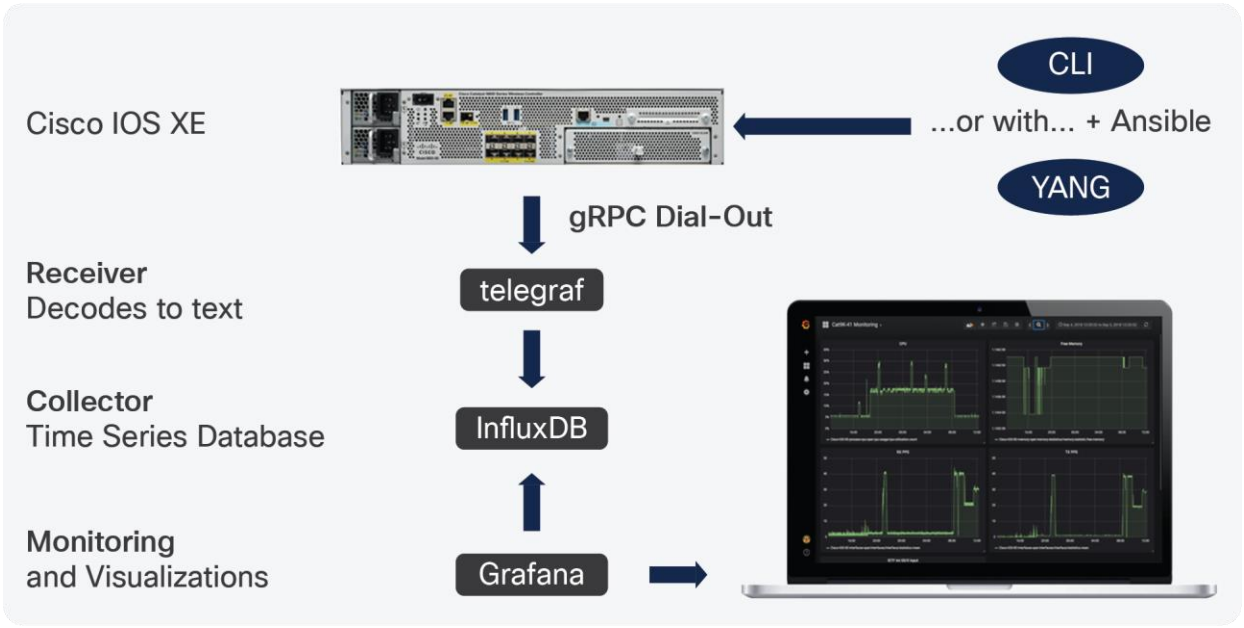


Figure 11.
gRPC workflow example

Verifying telemetry subscriptions

There are several show commands available to verify the status of the telemetry subscription configurations.

Examples of each are below:

```
show telemetry ietf subscription all
```

```
C9800-WLC-L-C#show telemetry ietf subscription all
  Telemetry subscription brief

  ID               Type        State       Filter type
  ---------------------------------------------------------
  1                Configured  Valid       xpath
  2                Configured  Valid       xpath
  3                Configured  Valid       xpath
  4                Configured  Valid       xpath
  5                Configured  Valid       xpath
  6                Configured  Valid       xpath
```

```
show telemetry ietf subscription <ID> detail
```

```
C9800-WLC-L-C#show telemetry ietf subscription 1 detail
Telemetry subscription detail:

  Subscription ID: 1
  Type: Configured
  State: Valid
  Stream: yang-push
  Filter:
    Filter type: xpath
    XPath: /wireless-client-oper:client-oper-data/dot11-oper-data
  Update policy:
    Update Trigger: periodic
    Period: 3000
  Encoding: encode-kvgpb
  Source VRF:
  Source Address: 10.1.1.5
  Notes:

  Receivers:
    Address                                 Port     Protocol        Protocol Profile
    ------------------------------------------------------------------------------------
    10.1.1.6                                57500    grpc-tcp
```

```
show telemetry ietf subscription <ID> receiver
```

```
jcohoe-cat9800#show telemetry ietf subscription 111 receiver
Telemetry subscription receivers detail:

  Subscription ID: 111
  Address: 10.85.134.66
  Port: 57500
  Protocol: grpc-tcp
  Profile:
  State: Connected
  Explanation:
```

gNMI

gNMI is a gRPC Network Management Interface. gNMI provides the mechanism to install, manipulate, and delete the configuration of network devices, and to view operational data. The content provided through gNMI can be modeled using YANG. gRPC is a remote procedure call developed by Google for low-latency, scalable distributions with mobile clients communicating to a cloud server. gRPC carries gNMI and provides the means to formulate and transmit data and operation requests.

gNMI configuration

To enable the gNMI interface enter the following commands:

```
configure terminal
gnxi
gnxi server
gnxi port 50052
exit
write memory
```

gNMI configuration Secure Mode

The following example shows how to enable the gNXI server secure mode.

```
configure terminal
gnxi
gnxi secure-trustpoint trustpoint1
gnxi secure-server
gnxi secure-client-auth
gnxi secure-port 9339
exit
write memory
```

Ensure the gNMI secure server is enabled.

```
show gnxi state detail
```

```
C9800-WLC-L-C#show gnxi state detail
Settings
========
  Server: Enabled
  Server port: 50052
  Secure server: Enabled
  Secure server port: 9339
  Secure client authentication: Enabled
  Secure trustpoint: trustpoint1
  Secure client trustpoint:
  Secure password authentication: Disabled
```

gNMI Operations

The following operations are supported over the gNMI interface:

gNMI GetRequest

gNMI SetRequest

gNMI Subscribe

Additional details on utilizing this interface are available from the Programmability Configuration Guide linked in the Additional Resources section.

Verifying gNMI status

To verify the gNMI interface is operational, run the command:

```
show gnxi state
```

Ensure the state is enabled and status is up.

```
C9800-WLC-L-C#show gnxi state
State              Status
-------------------------------
Enabled            Up
```

```
show gnxi state detail
```

```
C9800-WLC-L-C#show gnxi state detail
Settings
========
  Server: Enabled
  Server port: 50052
  Secure server: Disabled
  Secure server port: 9339
  Secure client authentication: Disabled
  Secure trustpoint:
  Secure client trustpoint:
  Secure password authentication: Disabled

GNMI
====
  Admin state: Enabled
  Oper status: Up
  State: Provisioned

  gRPC Server
  -----------
    Admin state: Enabled
    Oper status: Up

  Configuration service
  ---------------------
    Admin state: Enabled
    Oper status: Up

  Telemetry service
  -----------------
    Admin state: Enabled
    Oper status: Up
```

JSON IETF Encoding for YANG Data Trees

JSON is supported encoding for the gNMI interface. RFC 7951 defines JavaScript Object Notation (JSON) encoding for YANG data trees and their subtrees. gNMI uses JSON for encoding data in its content layer.

The JSON type indicates that the value is encoded as a JSON string. JSON_IETF-encoded data must conform to the rules for JSON serialization described in RFC 7951. Both the client and target must support JSON encoding.

Instances of YANG data nodes (leaves, containers, leaf-lists, lists, anydata nodes, and anyxml nodes) are encoded as members of a JSON object or name/value pairs. Encoding rules are identical for all types of data trees, such as configuration data, state data, parameters of RPC operations, actions, and notifications.

Every data node instance is encoded as a name/value pair where the name is formed from the data node identifier. The value depends on the category of the data node.

A leaf node has a value, but no children, in a data tree. A leaf instance is encoded as a name/value pair. This value can be a string, number, literal true or false, or the special array [null], depending on the type of the leaf. In the case that the data item at the specified path is a leaf node (which means it has no children, and an associated value) the value of that leaf is encoded directly. (A bare JSON value is included; it does not require a JSON object.)

The following example shows a leaf node definition:

```
leaf foo {
 type uint8;
}
```

The following is a valid JSON-encoded instance:

" foo" : 123

gNMI Wildcard

gNMI wildcard has been introduced in the 17.5 release. This is the ability to use a wildcard in a path to match multiple elements. Now, it is easier to know which variables are used, instead of having to perform a separate GET request.

There are two types of wildcards; implicit and explicit, and both are supported. Get paths support all types and combinations of path wildcards.

- Implicit wildcards: These expand a list of elements in an element tree. Implicit wildcard occurs when a key value is not provided for elements of a list.

  The following is a sample path implicit wildcard. This wildcard will return the descriptions of all interfaces on a device:

```
path {
  elem {
    name: "interfaces"
  }
  elem {
    name: "interface"
  }
  elem {
    name: "config"
  }
  elem {
    name: "description"
  }
}
```

Explicit wildcards: Provides the same functionality by specifying an asterisk (*) for either the path element name or key name.

The following sample shows a path asterisk wildcard as the key name. This wildcard returns the description for all interfaces on a device.

```
path {
  elem {
    name: "interfaces"
  }
  elem {
    name: "interface"
    key {
     key: "name"
      value: "*"
    }
  }
  elem {
    name: "config"
  }
  elem {
    name: "description"
  }
}
```

The following sample shows a path asterisk wildcard as the path name. This wildcard will return the description for all elements that are available in the Loopback111 interface.

```
path {
  elem {
    name: "interfaces"
  }
  elem {
    name: "interface"
    key {
      key: "name"
      value: "Loopback111"
    }
  }
   elem {
    name: "*"
  }
  elem {
    name: "description"
  }
}
```

Specifying an ellipsis (...) or a blank entry as element names. These wildcards can expand to multiple elements in a path.

The following sample shows a path ellipsis wildcard. This wildcard returns all description fields available under /interfaces.

```
path {
  elem {
    name: "interfaces"
  }
  elem {
    name: "..."
  }
  elem {
    name: "description"
  }
}
```

The following is a sample GetRequest with an implicit wildcard. This GetRequest will return the oper-status of all interfaces on a device.

```
path {
  elem {
    name: "interfaces"
    }
  elem {
    name: "interface"
  }
  elem {
    name: "state"
  }
  elem {
    name: "oper-status"
  }
},
type: 0,
encoding: 4
```

The following is a sample GetResponse with an implicit wildcard:

```
notification {
  timestamp: 1520627877608777450
  update {
    path {
      elem {
        name: "interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "\"FortyGigabitEthernet1/1/1\""
        }
      }
      elem {
        name: "state"
}
      elem {
        name: "oper-status"
      }
    }
    val {
      json_ietf_val: "\"LOWER_LAYER_DOWN\""
    }
  },

<snip>
…
</snip>

update {
    path {
      elem {
        name: "interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "\"Vlan1\""
```

```
        }
      }
      elem {
        name: "state"
      }
      elem {
        name: "oper-status"
      }
    }
    val {
      json_ietf_val: "\"DOWN\""
    }
  }
}
```

Verifying the Device Management Interface (DMI) processes are running

To verify the DMI processes, run the command:

```
show platform software yang-management process
```

```
C9800-WLC-L-C#show platform software yang-management process
confd               : Running
nesd                : Running
syncfd              : Running
ncsshd              : Running
dmiauthd            : Running
nginx               : Running
ndbmand             : Running
pubd                : Running
gnmib               : Running
```

Cisco YANG Suite

YANG Suite is a tool used for interacting with the NETCONF and RESTCONF gNMI interfaces. YANG Suite was publicly released onto GitHub. It can be used to generate and send XML RPC payloads to the device. YANG Suite provides a YANG API testing and validation environments that supports NETCONF, RESTCONF, gNMI, and gRPC Telemetry.

Figure 12.
Example of NETCONF operation in Cisco YANG Suite

YANG suite resources

- DevNet landing page: developer.cisco.com/yangsuite
- Documentation: developer.cisco.com/docs/yangsuite

YANG suite installation

Follow the instructions from the Github site at https://github.com/CiscoDevNet/yangsuite to complete the installation. Detailed instructions are available for Mac, Windows, and Linux. YANG Suite can be installed as a Docker Container or through Python package management. Installing YANG Suite as a Docker Container is the recommended install.

Prerequisites:

- Docker
- Docker Compose
- Docker Desktop for Mac
- Docker Desktop for Windows

For Mac and Windows follow these instructions to install Docker Compose.

For Linux follow these instructions to install Docker Compose.

Quick start

1. Clone the repository https://github.com/CiscoDevNet/yangsuite
2. Run **start_yang_suite**.sh or,
3. Run **docker-compose up** if you have already ran the start_yang_suite.sh
4. Access the YANG Suite tool at https://localhost:8443

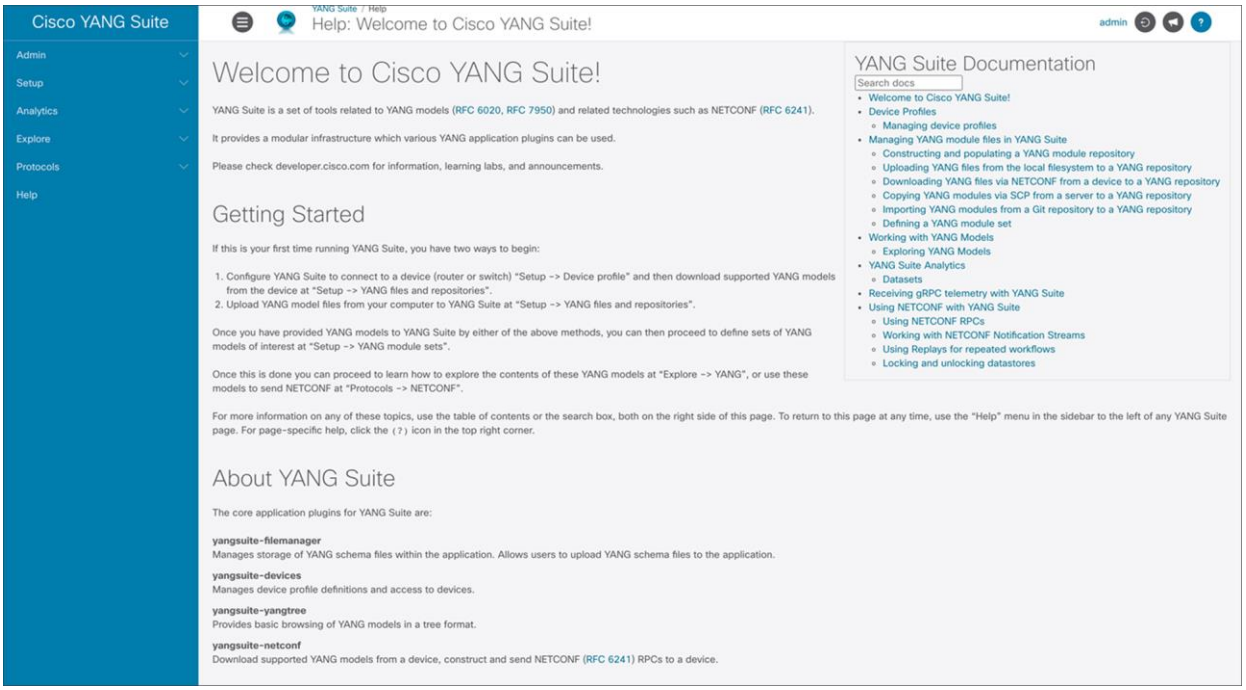
```
git clone https://github.com/CiscoDevNet/yangsuite
cd yangsuite/docker/ ; ./start_yang_suite.sh
or
cd yangsuite/docker/ ; docker compose up
```

When the YANG Suite is ready for use, the following will be seen:

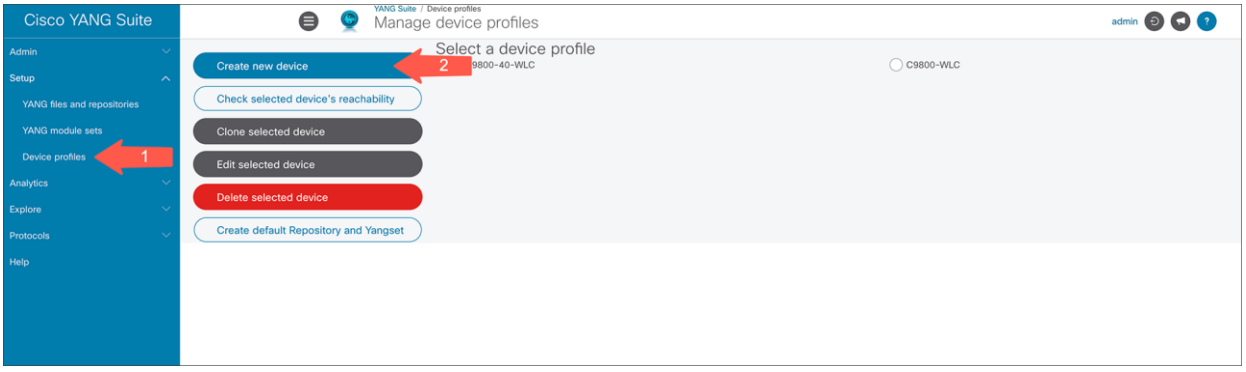
```
yangsuite_1  | spawned uWSGI master process (pid: 34)
yangsuite_1  | spawned uWSGI worker 1 (pid: 38, cores: 1)
yangsuite_1  | spawned uWSGI worker 2 (pid: 39, cores: 1)
yangsuite_1  | spawned uWSGI worker 3 (pid: 40, cores: 1)
yangsuite_1  | spawned uWSGI worker 4 (pid: 41, cores: 1)
yangsuite_1  | spawned uWSGI worker 5 (pid: 42, cores: 1)
```

YANG Suite Day 0 configuration

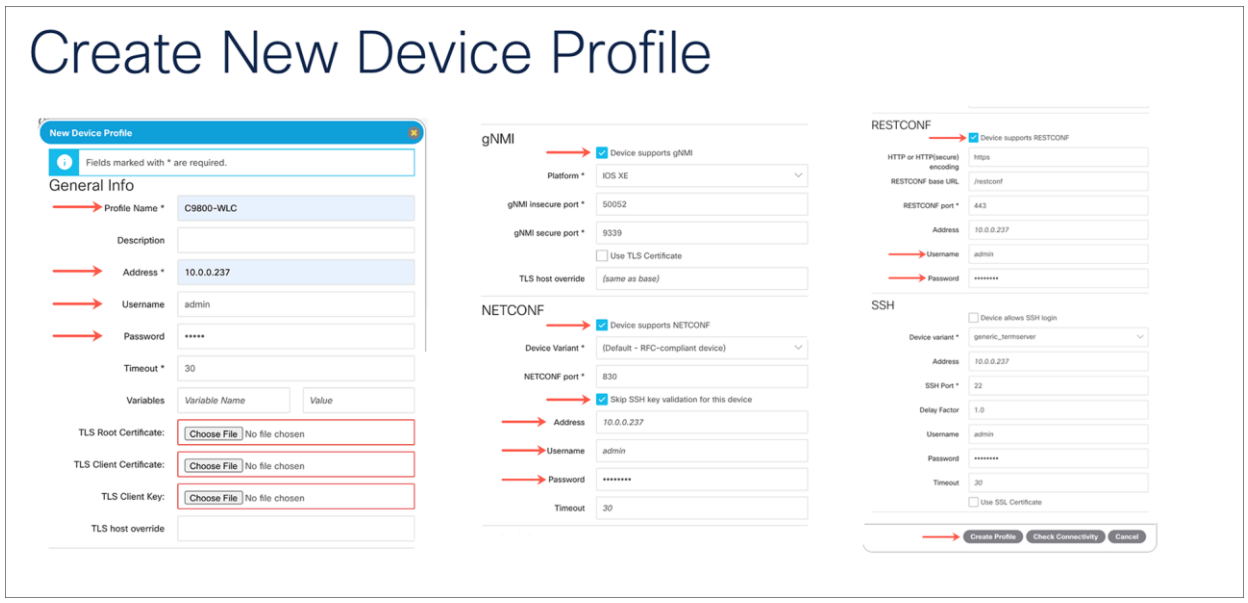
When YANG Suite is installed and running, a few tasks must be completed before interacting with the Catalyst 9800. Navigate to the Google/Firefox web browser and access YANG Suite at http://localhost:8443. Log in using the credentials configured during the installation process. Once logged in, you'll end up at the main YANG Suite application window.



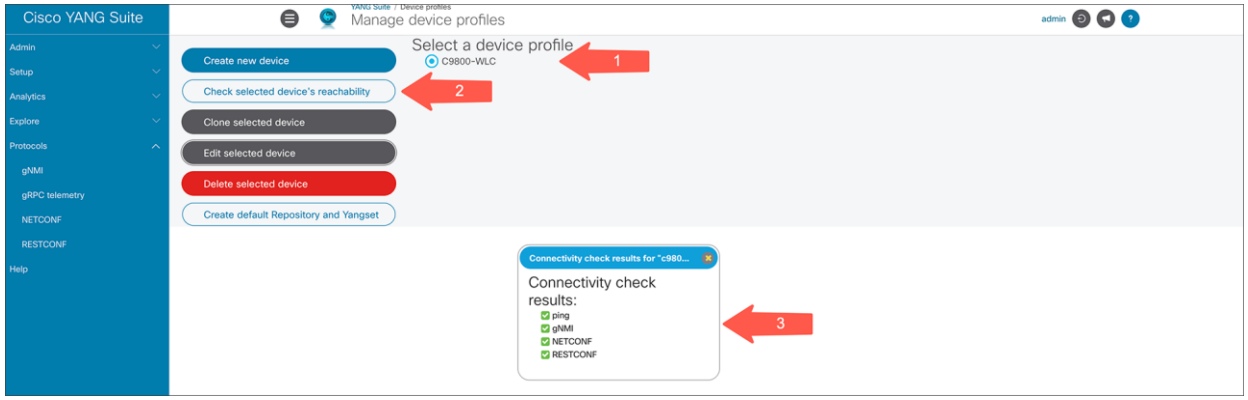
Navigate to the **Setup > Device profiles** menu. Click **Create new device**.



The New Device Profile window will pop up where information about the device must be added. Enter the profile **name, address, username,** and **password**. Since YANG Suite now supports gNMI, check the **Device supports gNMI** box. Check the **Device supports NETCONF, Skip SSH key validation for this device**, and **Device supports RESTCONF** boxes. Make sure to enter **username** and **password** for both NETCONF and RESTCONF, if you have configured a separate authentication. Click **Create Profile** to add the device to YANG Suite.



Once the device is added, check the device reachability by clicking **Check selected device's reachability**. Make sure to see the green checkmark for ping, gNMI, NETCONF, and RESTCONF.

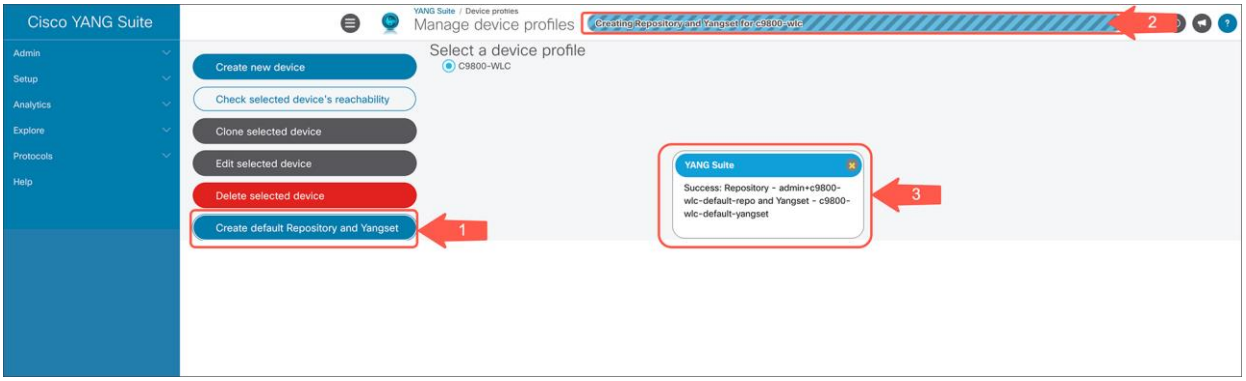


Creating default repository and YANG set

From within the YANG Suite, the YANG modules can be downloaded from the device. The Yang repository can be created automatically from the Manage Device Profiles page. Yang Set is a subset of a YANG repository that consists of a set of modules and any other necessary dependencies. A YANG set could store an entire repository, but it's more efficient to narrow the set down to only the models we are interested in. A YANG Set needs to be created to Build and Run RPC(s).

Follow the steps to download the models:

On the Manage Device Profiles page, click the **Create default Repository and Yangset** button. The following option will automatically create a default repository and a default set.

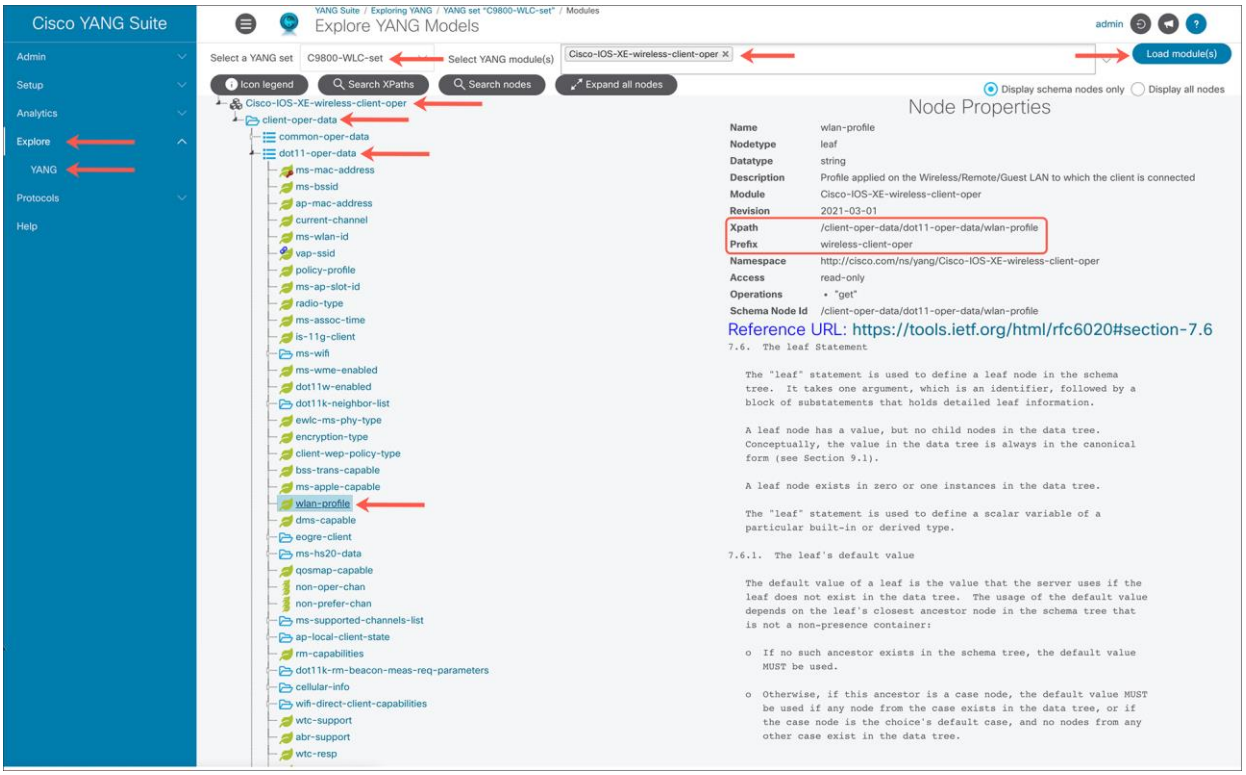


When the download is completed, the desired modules will appear in the repository and set boxes.

Explore YANG models

With the created YANG set we can easily explore the YANG data models. From the menu on the left side of the page, select **Explore > YANG**.

1. From the left navigation pane select **Explore > YANG**.
2. From the **Select a YANG set** drop-down menu, select a newly created set.
3. In the **Select YANG modules(s)** box, enter any wireless data model of choice. In this example, we explore the **Cisco-IOS-XE-wireless-client-oper** module.
4. Click the **Load module(s)**. After a moment, the left column will be populated with a tree view of the contents of the module. Initially the tree view shows only the module itself, but you can click the triangle icon next to it to expand the tree.
5. Refer to the screenshot to examine the structure of the model and its content.



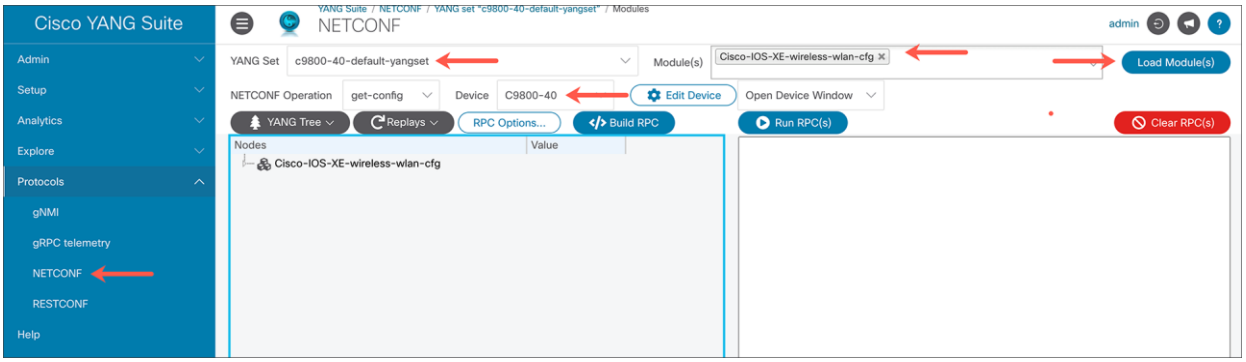
Two important pieces of YANG model metadata are the Xpath and the Prefix. These fields are used with Model-Driven Telemetry to retrieve information. If a telemetry subscription was to be created based on the IOS XE interfaces YANG data model, the Xpath of "/interfaces/interface/interface-type" and "interfaces-ios-xe-oper" would be used to retrieve and publish information from those models.

NETCONF plus YANG Suite

YANG Suite enables interaction with the devices using most of the programmatic interfaces: NETCONF, RESTCONF, gNMI, and gRPC. The example below shows the use of a NETCONF programmatic interface on the C9800 Wireless controller. With the help of the NETCONF Operation get-config, it's easy to retrieve all or part of the specified wireless configuration datastore. Also, the NETCONF Operation edit-config loads all of a specified configuration to a specified target configuration.

Steps to access the NETCONF plugin using YANG Suite:

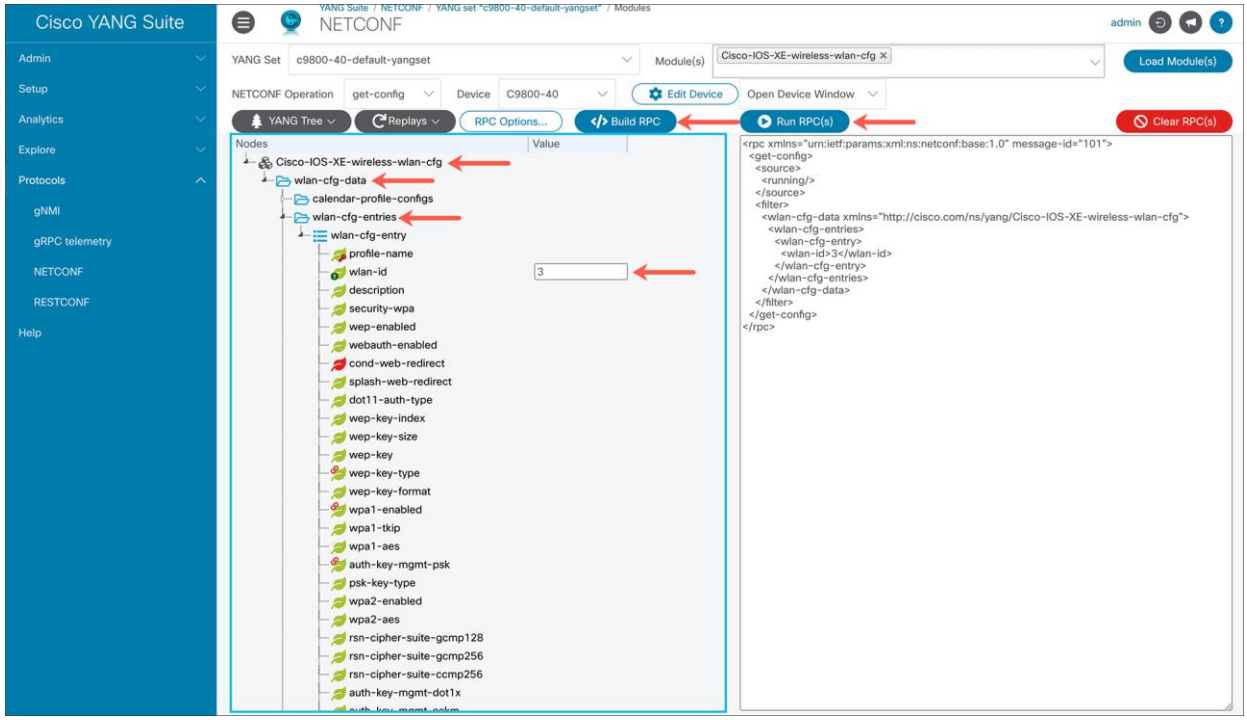
1. From the left navigation pane select **Protocols > NETCONF**.
2. Select the created **YANG Set**.
3. Select the Module(s): **Cisco-IOS-XE-wireless-wlan-cfg**.
4. Select a C9800 device from a drop-down menu, and click **Load Module(s).**



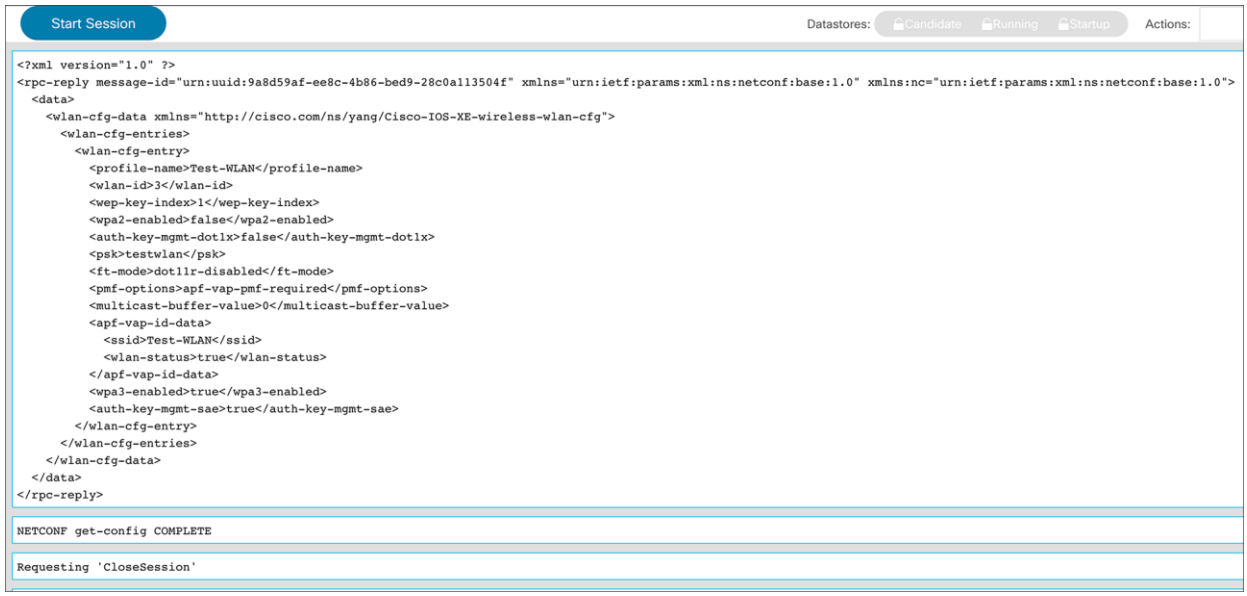
The C9800 Wireless Controller from the above example has been preconfigured with a wlan-id 3. To retrieve the part of WLAN configuration datastore, a get-config NETCONF operation is used.

Steps to build and run RPC(s):

1. Expand the **Cisco-IOS-XE-wireless-wlan-cfg, wlan-cfg-data**, and **wlan-cfg-entries** nodes tree.
2. In the value column next to the wlan-id, type the number of the WLAN you wish to retrieve.
3. Click the **Build RPC** button, and the RPC appears in the text window.
4. Click RUN RPC(s) button, and a new dialog window appears in a separate tab.



The configuration of the WLAN should be displayed if everything is done correctly.



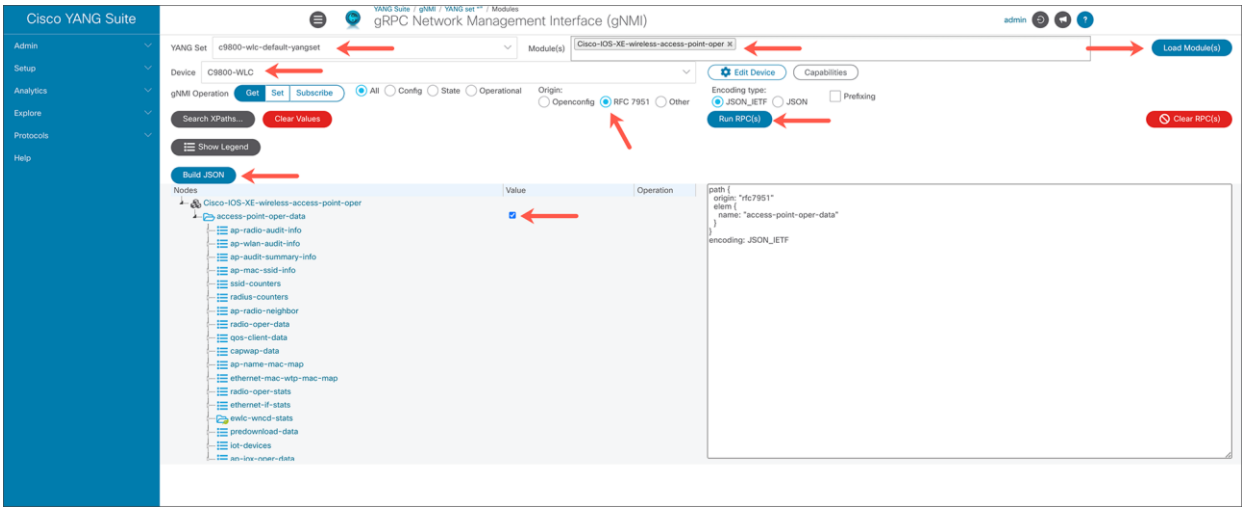
gNMI plus YANG Suite

The gNMI tooling can be used to retrieve information from the device. With the help of programmatic interfaces and YANG Suite, it became easier to retrieve operational information from the IOS XE devices. YANG Suite provides a YANG API testing and validation environment that supports gNMI. The gNMI tooling uses JSON_IETF for encoding data in its content layer. In this example, we run RPC for one of the wireless modules **`Cisco-IOS-XE-wireless-access-point-oper`**.

Steps to access the gNMI plugin using YANG Suite:

1. From the left navigation pane, select **Protocols > gNMI**.
2. Select the created **YANG Set**.
3. Select the Module(s): **Cisco-IOS-XE-wireless-access-point-oper.** Click **Load Modules**.
4. From the drop-down list, select the Device.
5. For Origin, select **RFC7951**.

In the Nodes section, make sure to expand the loaded YANG data model. Click the **Value** column next to the **access-point-oper-data** row. To narrow down the search, each individual node can be selected from the list. To generate the payload, click the **Build JSON** button. Once the payload is generated, it is ready to Run RPC.



The gNMI GET, gNMI GET Response, and gNMI Response value decoded will show up in separate window.

```
gNMI GET
========
path {
  origin: "rfc7951"
  elem {
    name: "access-point-oper-data"
  }
}
encoding: JSON_IETF
```

```
gNMI GET Response
=================
notification {
  timestamp: 1628593197288836770
  update {
    path {
      origin: "rfc7951"
      elem {
        name: "Cisco-IOS-XE-wireless-access-point-oper:access-point-oper-data"
      }
    }
    val {
      json_ietf_val: "{\"ap-radio-neighbor\":[{\"ap-mac\":\"04:5f:b9:1f:0b:40\",\"slot-id\":0,\"bssid\":\"0c:7c:28:76:9b:34\",\"ssid\":\"
    }
  }
}
```

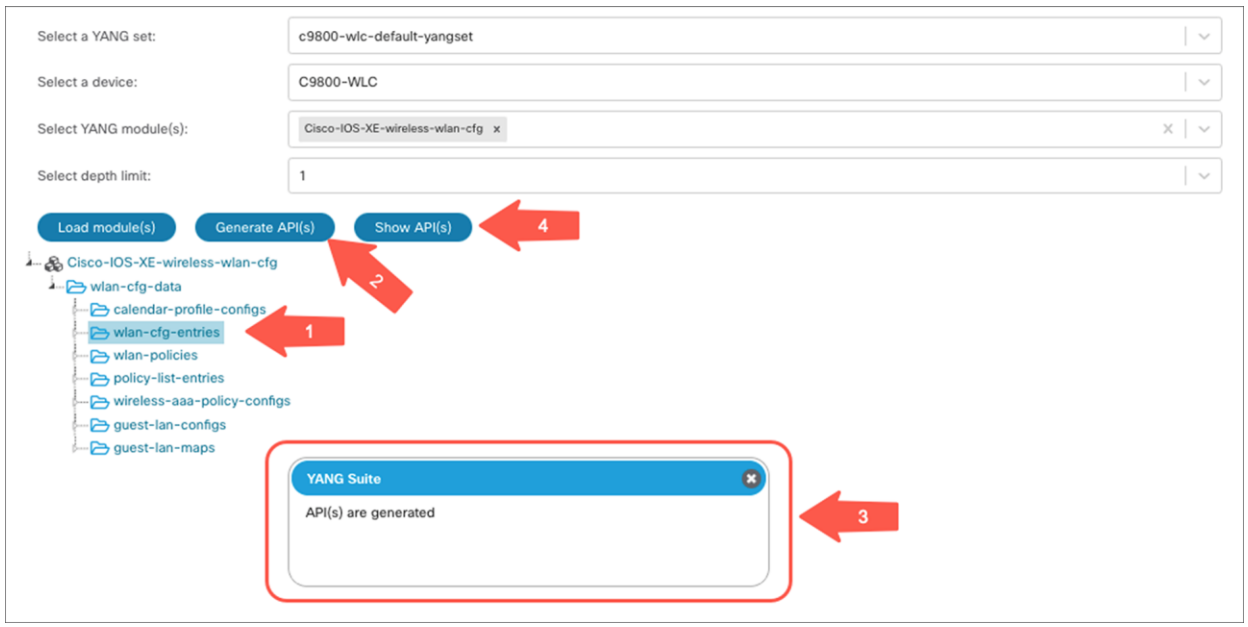
```
gNMI Response value decoded
===========================
```

```
[
  {
    "ap-radio-neighbor": [
      {
        "ap-mac": "                ",
        "slot-id": 0,
        "bssid": "0c:7c:28:76:9b:34",
        "ssid": "TrickRoom_5GHz",
        "rssi": -74,
        "channel": 0,
        "primary-channel": 11,
        "last-update-rcvd": "2021-08-10T10:59:32.016979+00:00"
      },
      '
```

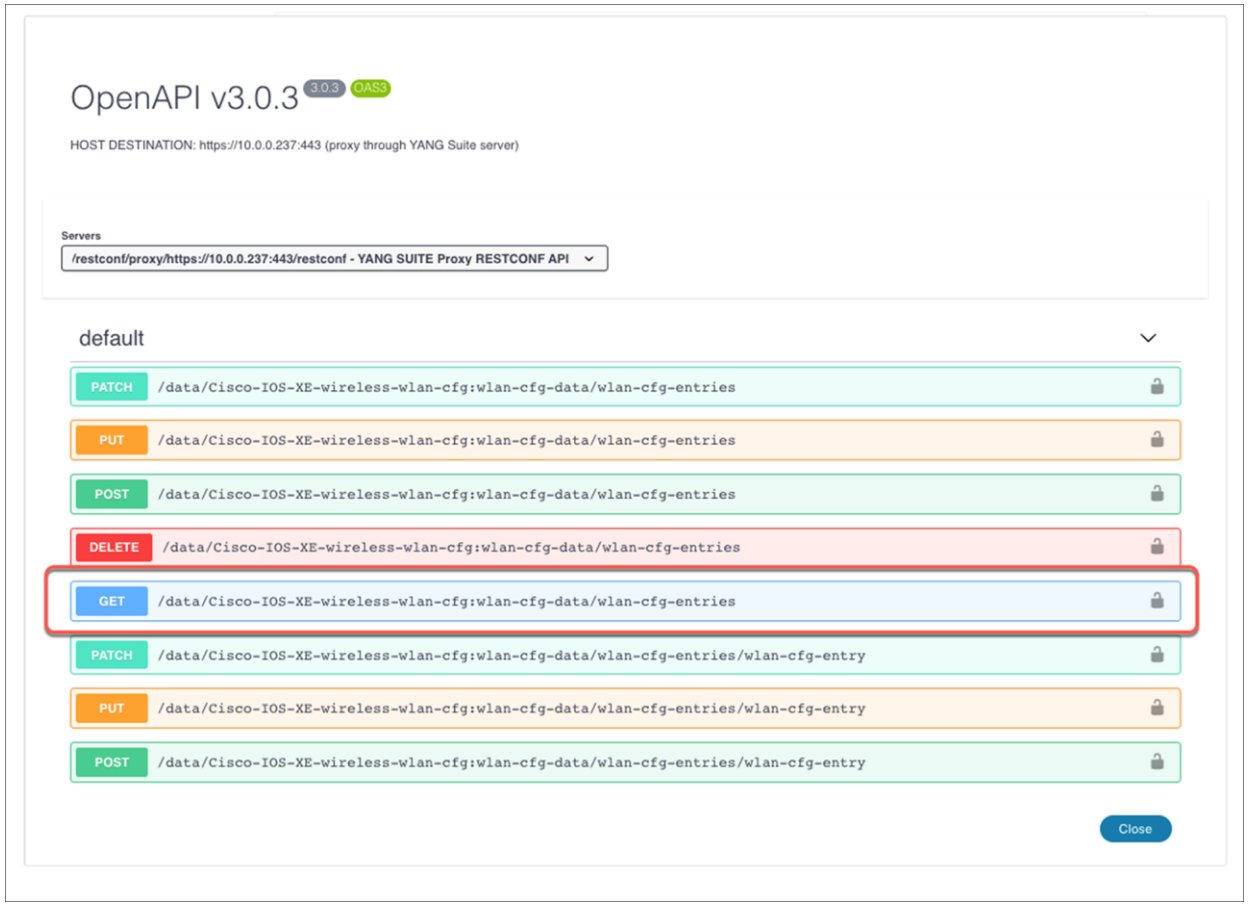
RESTCONF plus YANG Suite

To access the RESTCONF plugin using YANG Suite, navigate to the **Protocols > RESTCONF** on the left pane of the YANG Suite application. Make sure to fill out all the necessary fields to load the YANG modules from the device and generate API(s). For the YANG module, we used the `Cisco-IOS-XE-wireless-wlan-cfg` to get the information about the configured WLANs on the C9800 device.

When these four fields are correctly set, click **Load module(s)** to generate the tree.

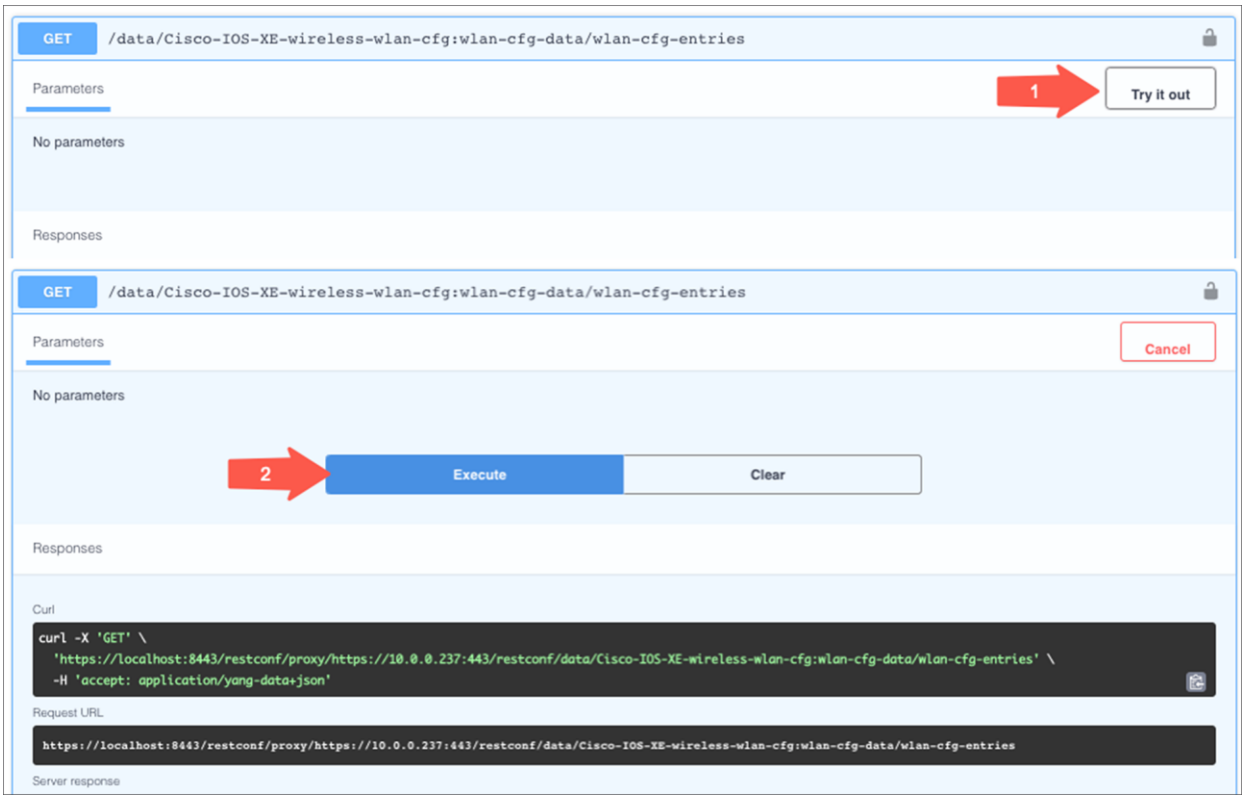


When the module is loaded, expand the tree. Select the **wlan-cfg-entries container**, then click the **Generate API(s)** button. The message from YANG Suite outlined in red above indicates that the API(s) were successfully generated and ready to be viewed. When you see a similar pop-up message, close it. Click the Show API(s) button to view all the available requests.

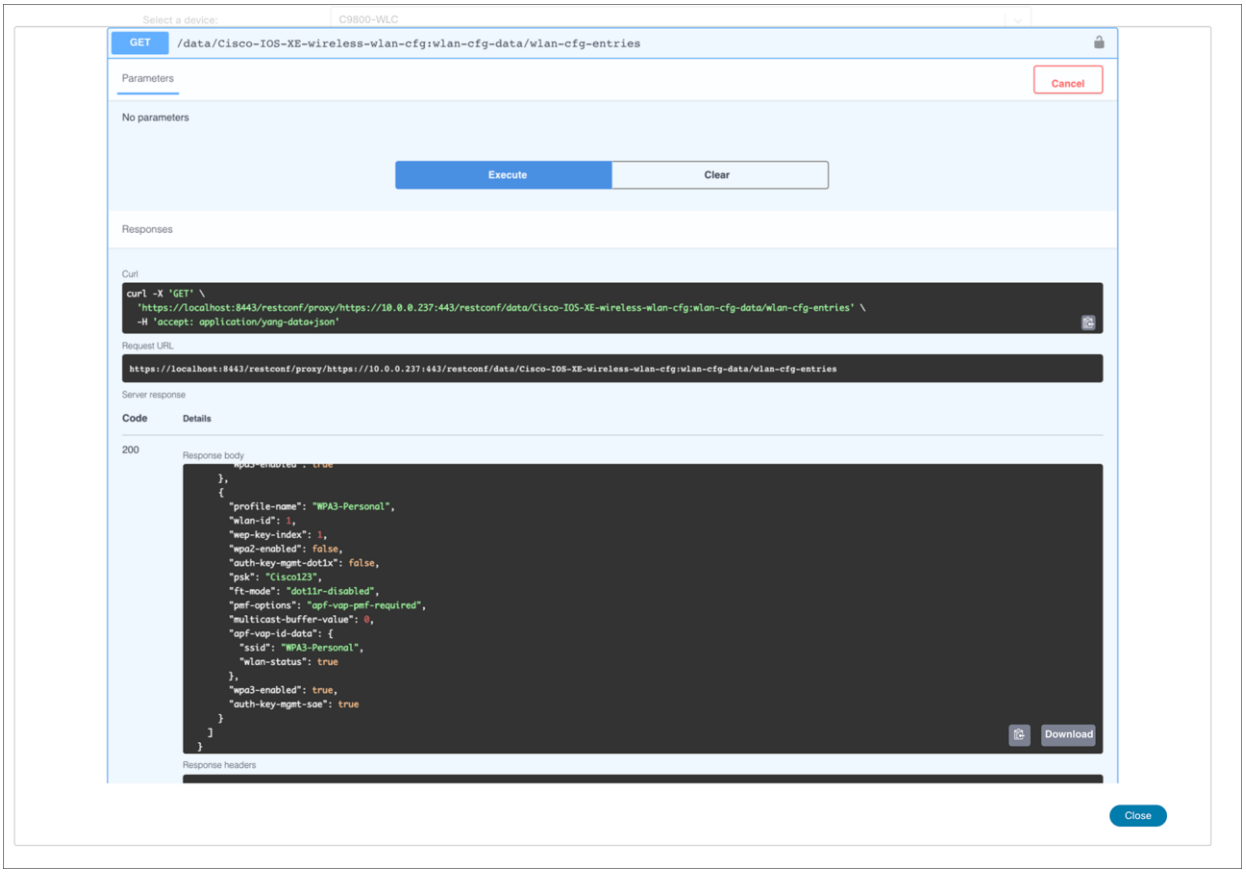


All the default requests are displayed on a separate page. In the following example, we take a closer look at the GET request. If you click any of the above requests, you'll be able to expand the data and run RPC.

Select the **GET** request. Click the **Try it out button**, and then click the **Execute** button. The RESTCONF API call will be sent to the device, which will reply with the WLAN config data as requested.



If the request is successful, a response of **200** will be returned with the JSON response body. If the response body doesn't contain any information, a response of **204** will be returned, which also indicates a successful request.



Ansible automation

Ansible is a popular and easy to use open-source software suite that automates software provisioning, configuration, and management. It connects to and controls devices via SSH, NETCONF, and a variety of other protocols as well. Ansible is agentless, meaning there are no installation or requirements on the target device, other than having an accessible API or interface. It is minimal in nature and provides a secure and reliable way to interact with remote devices. It is commonly used with other automation tools to accomplish complex workflows as it is highly adaptable. Below are some examples of using Ansible to complete basic Day 0 configuration tasks.

Ansible has several components that work together to provide a holistic solution. There are modules that are reusable and a standalone script that can be called. Tasks call upon modules to perform an action. When there are multiple tasks, a play can be used to call the tasks in order. A playbook is then used when there are multiple plays. Finally, a role is a set of playbooks.

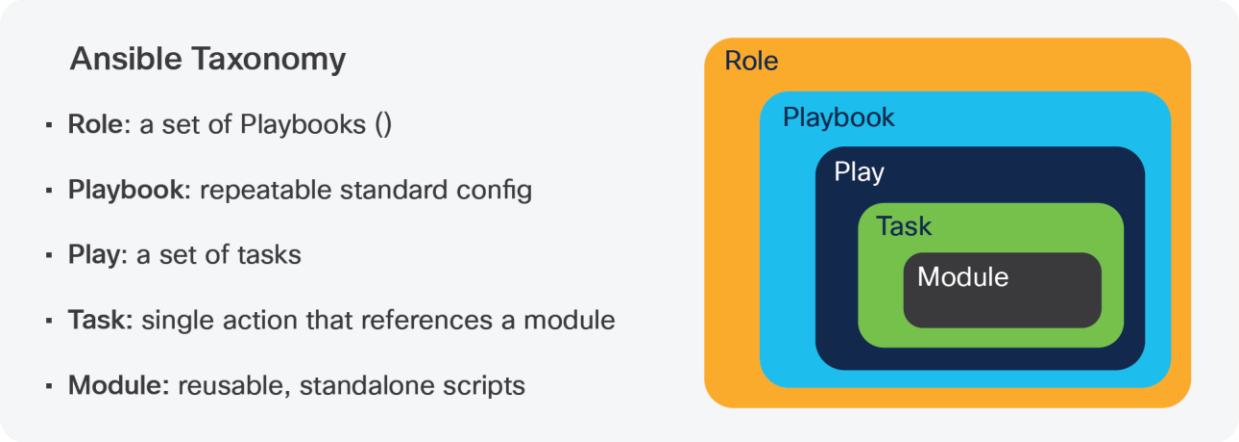


Figure 13.
Ansible Taxonomy

Ansible Example: Enable NETCONF-YANG and RESTCONF

The hosts file has connection details and device-specific information including credentials. In this example, it is assumed that the Catalyst 9800 has already been configured to allow SSH logins, and that the enable password has also been set. The hosts files and YAML file are used together to accomplish a task. In this case, the tasks are to connect to the CLI over SSH, enter enable mode, and execute the required IOS commands to enable netconf-yang and set up the AAA requirements.

Ansible hosts file for Day 0 configuration

This example **hosts.txt** file contains the variables needed to successfully establish a connection to the device.

```
[C9800]
10.0.0.237

[ios_xe:children]
C9800

[ios_xe:vars]
ansible_connection=network_cli
ansible_network_os=ios
ansible_password=cisco
ansible_python_interpreter = "/usr/bin/python"
```

Ansible YAML configuration file to enable NETCONF and RESTCONF

This example enable_netconf_yang.yaml YAML file that can be used to enable the NETCONF interface on the device and configure the authentication prerequisites including adding a user.

```
---
- hosts: C9800
  gather_facts: no

  tasks:
   - ios_config:
      commands:
       - aaa new-model
       - aaa authorization exec default local
       - aaa authentication login default local
       - username netconf privilege 15 password 0 netconf
       - netconf-yang
       - ip http secure-server
       - restconf
      save_when: modified
```

Ansible YAML CLI automation show file to verify NETCONF and RESTCONF configuration

This example **cat9800_verify.yaml** YAML file will run two IOS XE SHOW commands to verify that netconf-yang and restconf is enabled. and The output will be registered and displayed on the screen when executed:

```
---
- hosts: C9800
  gather_facts: no

  tasks:
   - ios_command:
      commands:
      - show run | i netconf-yang
      - show run | i restconf
     register: show
   - debug: var=show.stdout_lines
```

Executing the task to enable and verify NETCONF and RESTCONF

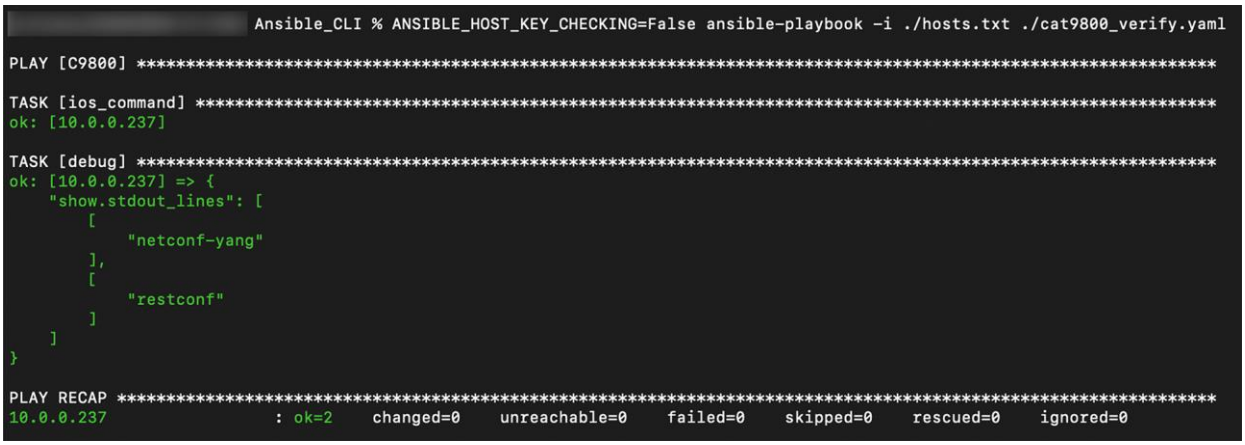
The ansible-playbook command can be used to execute the task that we define above to enable the NETCONF-YANG interface on the device. In this example, we define a variable to set the Host Key Checking to false, so that the SSH host key is not validated. In production environments, it is important to verify the authenticity of the device being accessed. However in this lab example, the check is set to False for ease of use.

From the command line, execute the following command to run the enable_netconf_yang.yaml configuration:

```
$ ANSIBLE_HOST_KEY_CHECKING=False ansible-playbook -i ./hosts.txt
./enable_netconf_yang.yaml
```

From the command line, execute the following command to run the cat9800_verify.yaml configuration

```
$ ANSIBLE_HOST_KEY_CHECKING=False ansible-playbook -i ./hosts.txt ./cat9800_verify.yaml
```



Ansible Example: Use NETCONF to create a WLAN

Previously we have learned how to create a WLAN with the help of NETCONF. Now, let's create a new WLAN with the help of Ansible + NETCONF. This example sends xml into the NETCONF interface to add a WLAN. Create a folder named **ansible**. In the newly created folder, create three files: **host.txt, ansible.cfg,** and **add_wlan.yaml.** the content for each file is listed below. Make sure to change the IP address, username, and password.

The host.txt file:

```
[C9800]
172.20.229.206

[ios_xe:children]
C9800
[ios_xe:vars]
ansible_connection=netconf
ansible_network_os=ios
ansible_password=netconf
ansible_python_interpreter = "/usr/bin/python"
```

ansible.cfg:

```
[defaults]
inventory = ./hosts.txt
host_key_checking = False
roles_path     = ./
remote_user = admin
```

add_wlan.yaml

```
- hosts: C9800
  gather_facts: no
  connection: netconf
  remote_user: netconf

  tasks:
  - name: establish subscription
    netconf_config:
      content: |
          <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
            <wlan-cfg-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-wireless-wlan-cfg">
              <wlan-cfg-entries>
                <wlan-cfg-entry>
                  <profile-name>Test</profile-name>
                  <wlan-id>5</wlan-id>
                  <security-wpa>true</security-wpa>
                  <wpa2-enabled>true</wpa2-enabled>
                  <apf-vap-id-data>
                    <broadcast-ssid>true</broadcast-ssid>
                    <ssid>Test-WLAN</ssid>
                  </apf-vap-id-data>
                </wlan-cfg-entry>
              </wlan-cfg-entries>
            </wlan-cfg-data>
          </config>
```

From the command line, execute the following command to run the add-wlan.yaml configuration.

```
ansible-playbook add_wlan.yaml
```

```
annkomar@ANNKOMAR-M-910H ansible % ansible-playbook  add_wlan.yaml

PLAY [C9800] ***********************************************************************************************************

TASK [establish subscription] ******************************************************************************************
ok: [172.20.229.206]

PLAY RECAP *************************************************************************************************************
172.20.229.206             : ok=1    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Guest Shell

Guest Shell is a virtualized Linux-based environment, designed to run custom Linux applications, including Python for automated control and management of Cisco devices. Using the Guest Shell, you can also install, update, and operate third-party Linux applications. The guest shell is bundled with the system image and can be installed using the guestshell enable Cisco IOS command. This container shell provides a secure environment, decoupled from the host device, in which users can install scripts or software packages and run them. The existing network hardware is used to deliver the scalability, high availability, and flexibility required, with no requirements for dedicated or separate compute. The Guest Shell environment is intended for tools, Linux utilities, and manageability rather than networking.

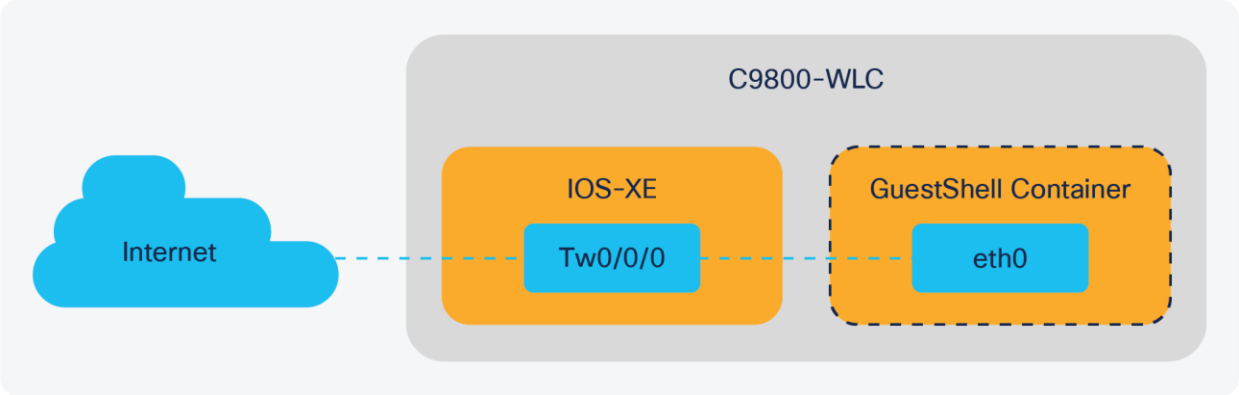


Figure 14.
Guestshell on C9800-WLC

Enabling and Running the Guest Shell

Before enabling the Guest Shell, IOx must be configured. If IOx is not configured, a message to configure IOx is displayed. Removing IOx removes access to the Guest Shell. To enable and operate Guest Shell, management interface needs to be configured on C9800. To enable IOx, enter the following commands:

```
configure terminal
iox
exit
```

Enabling Guest Shell on the Management Interface

```
configure terminal
app-hosting appid <name>
app-vnic management guest-interface <interface number>
end
show app-hosting list
```

Once the prerequisite configuration is setup then enable and enter the Guest Shell container:

```
guestshell enable
```

```
Device#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Device(config)#app-hosting appid guestshell
Device(config-app-hosting)#app-vnic management guest-interface 1
Device(config-app-hosting-mgmt-gateway)#end
Device#show app-hosting list
App id                                   State
---------------------------------------------------------
guestshell                               DEPLOYED

Device#guestshell enable
Interface will be selected if configured in app-hosting
Please wait for completion
guestshell activated successfully
Current state is: ACTIVATED
guestshell started successfully
Current state is: RUNNING
Guestshell enabled successfully

Device#show app-hosting list
App id                                   State
---------------------------------------------------------
guestshell                               RUNNING
```

Verifying IOx Status

To confirm that the IOX service has been enabled, enter the **`show iox-service`** command and ensure IOx Cisco application hosting framework (CAF), IOx service IOxman, and Libertd are in Running state.

```
Device#show iox-service

IOx Infrastructure Summary:
---------------------------
IOx service (CAF)              : Running
IOx service (HA)               : Not Supported
IOx service (IOxman)           : Running
IOx service (Sec storage)      : Not Supported
Libvirtd 5.5.0                 : Running
```

```
Device#show app-hosting list

App id                                   State
---------------------------------------------------------
guestshell                               RUNNING
```

Accessing and Using Guest Shell

Linux commands can be run directly from the IOS CLI. The **guestshell** run bash command opens the Guest Shell bash prompt. To log into guest shell run the following command:

```
C9800-40#guestshell run bash
```

Guest Shell Usage

From the Guest Shell prompt, you can run Linux commands:

```
C9800-40 #guestshell run bash
[guestshell@guestshell ~]$ pwd
/home/guestshell
[guestshell@guestshell ~]$ whoami
guestshell
[guestshell@guestshell ~]$ uname -a
Linux guestshell 5.4.69 #1 SMP Fri Mar 19 21:47:56 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
```

Accessing the Python Interpreter

Python scripts can be run in the Guest Shell. The **guestshell run python3** commands launch the Python Interpreter.

```
guestshell run python3
```

Accessing the IOS CLI from the Guest Shell

The **dohost** command is built into Guest Shell and will send the command directly to the device. The command is limited to exec privilege mode and won't work for the config mode.

```
[[guestshell@guestshell ~]$ dohost 'show ip int brief'
 Interface              IP-Address      OK? Method Status                Protocol
 Tw0/0/0                unassigned      YES unset  down                  down
 Tw0/0/1                unassigned      YES unset  down                  down
 Tw0/0/2                unassigned      YES unset  down                  down
 Tw0/0/3                unassigned      YES unset  down                  down
 Te0/1/0                unassigned      YES unset  up                    up
 Te0/1/1                unassigned      YES unset  up                    up
 GigabitEthernet0       unassigned      YES NVRAM  down                  down
 Vlan1                  10.0.0.237      YES NVRAM  up                    up
 Vlan100                20.0.0.20       YES NVRAM  up                    up
```

Note: The **dohost** command requires the ip hhtp server command to be configured on the device

Once the interactive shell is entered, the **clip** python module can be used to execute

IOS CLI commands:

```
>>> from cli import clip
>>> clip("show wlan summary")
```

```
C9800-40#guestshell run python3
Python 3.6.8 (default, Dec 22 2020, 19:04:08)
[GCC 8.4.1 20200928 (Red Hat 8.4.1-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from cli import clip
>>> clip("show wlan summary")
Number of WLANs: 3
ID   Profile Name                    SSID                            Status Security

-------------------------------------------------------------------------------------------------
-----------------------------------------
1    WSIM                            WSIM                            UP     [open]

2    WSIM_PSK                        WSIM_PSK                        UP     [WPA2][PSK][AES]

3    Test-WLAN                       Test-WLAN                       UP     [WPA3][SAE][AES]
```

Disabling and Destroying the Guest Shell

The **`guestshell disable`** command shuts down and disables Guest Shell.

The **`guestshell destroy`** command removes the rootfs from the flash filesystem.

Non-Interactive Python

Guest Shell can execute python scripts in a non-interactive environment. Copy the python script from the TFTP server to the bootflash, then execute it with the **guestshell run python3** command. Alternately, use the 'vi' editor within the Bash environment to create the python file.

A very easy python script can be executed using the **guestshell run python3** command. Copy the whole script shown below to the **/bootflash/guest-share** location on your device. You can modify the script to your own needs. The purpose of this section is to showcase the guestshell capability:

```
#!/usr/bin/python


from cli import clip
clip("show wlan summary")
exit()
```

The command to execute the python script is:

```
guestshell run python3 /bootflash/show_wlans.py
```

```
C9800-WLC-L-C#guestshell run python3 /bootflash/guest-share/show_wlans.py
Number of WLANs: 3
ID   Profile Name                  SSID                          Status Security

------------------------------------------------------------------------------------------------
------
1    WPA3-Personal                 WPA3                          UP     [WPA3][802.1x][AES]

2    Home-Kit                      Home-Kit                      UP     [WPA2 + WPA3][802.1x][AES]

3    OpenRoaming                   OpenRoaming                   UP     [WPA2 + WPA3][802.1x][AES]
```

Guest Shell with EEM

Embedded Event Manager (EEM) is a distributed and customized approach to event detection and recovery offered directly in a Cisco IOS device. EEM offers the ability to monitor events and take informational, corrective, or any desired EEM action when the monitored events occur or when a threshold is reached. An EEM policy is an entity that defines an event and the actions to be taken when that event occurs.

The Embedded Event Manager (EEM) can be used to execute Python scripts within the Guest Shell environment. In this example, whenever a syslog message is generated indicating an AP joined or disjoined the controller, then the guestshell_script.py Python script will run. This script runs some CLI commands and saves the output to the bootflash.

Configure EEM with the following commands so whenever a syslog message with “Joined” or “Disjoined” is generated, a Python script will be executed. Ensure the Python file name and path is correct, and that the Python script can run successfully within Guest Shell prior to configuring EEM.

```
configure terminal
event manager applet ap_join
event syslog pattern “Joined|Disjoined”
action 0.0 cli command “en”
action 0.1 syslog msg “AP Join or Disjoin detected – starting show_ap script”
action 0.2 cli command “guestshell run python /bootflash/show_ap.py”
action 0.3 syslog msg “AP Join or Disjoin script completed”
```

```
Enter configuration commands, one per line.  End with CNTL/Z.
C9800-WLC-L-C(config)#event manager applet ap_join
C9800-WLC-L-C(config-applet)#event syslog pattern "Joined|Disjoined"
C9800-WLC-L-C(config-applet)#action 0.0 cli command "en"
C9800-WLC-L-C(config-applet)#action 0.1 syslog msg "AP Join or Disjoined detected - starting show_ap script"
C9800-WLC-L-C(config-applet)#action 0.2 cli command "guestshell run python /bootflash/show_ap.py"
C9800-WLC-L-C(config-applet)#action 0.3 syslog msg "AP Join or Disjoin script completed"
C9800-WLC-L-C(config-applet)#end
C9800-WLC-L-C#wr
```

Guest Shell Resources

Resources used by the Guest Shell container can be checked with the CLI command. The hardware resource allocations for CPU, memory and disk are displayed.

```
show app-hosting utilization appid guestshell
```

```
C9800-WLC-L-C#show app-hosting utilization appid guestshell
Application: guestshell
CPU Utilization:
  CPU Allocation: 800 units
  CPU Used:       0.00 %
  CPU Cores:      8-10

Memory Utilization:
  Memory Allocation: 256 MB
  Memory Used:       85952 KB
Disk Utilization:
  Disk Allocation: 1 MB
  Disk Used:       0.00 MB
```

Conclusion

The Cisco IOS XE network OS delivers an innovative level of programmability and automation, decreasing the complexity of the business and network. Now, we understand the need of programmable interfaces and the difference between them. Using the details within the guide the programmatic interfaces can be enabled and configured for use to communicate with the Cisco devices. Configured and Dynamic telemetry subscriptions can be established using open-source tools. Example XML payloads can be used to create, verify, and remove a WLAN programmatically using the YANG Suite tools. An example Ansible configuration can be used to enable the programmatic interfaces on the Catalyst 9800 device.

Additional resources

Cisco IOS XE Programmability Book

https://www.cisco.com/c/dam/en/us/products/collateral/enterprise-networks/nb-06-ios-xe-prog-ebook-cte-en.pdf

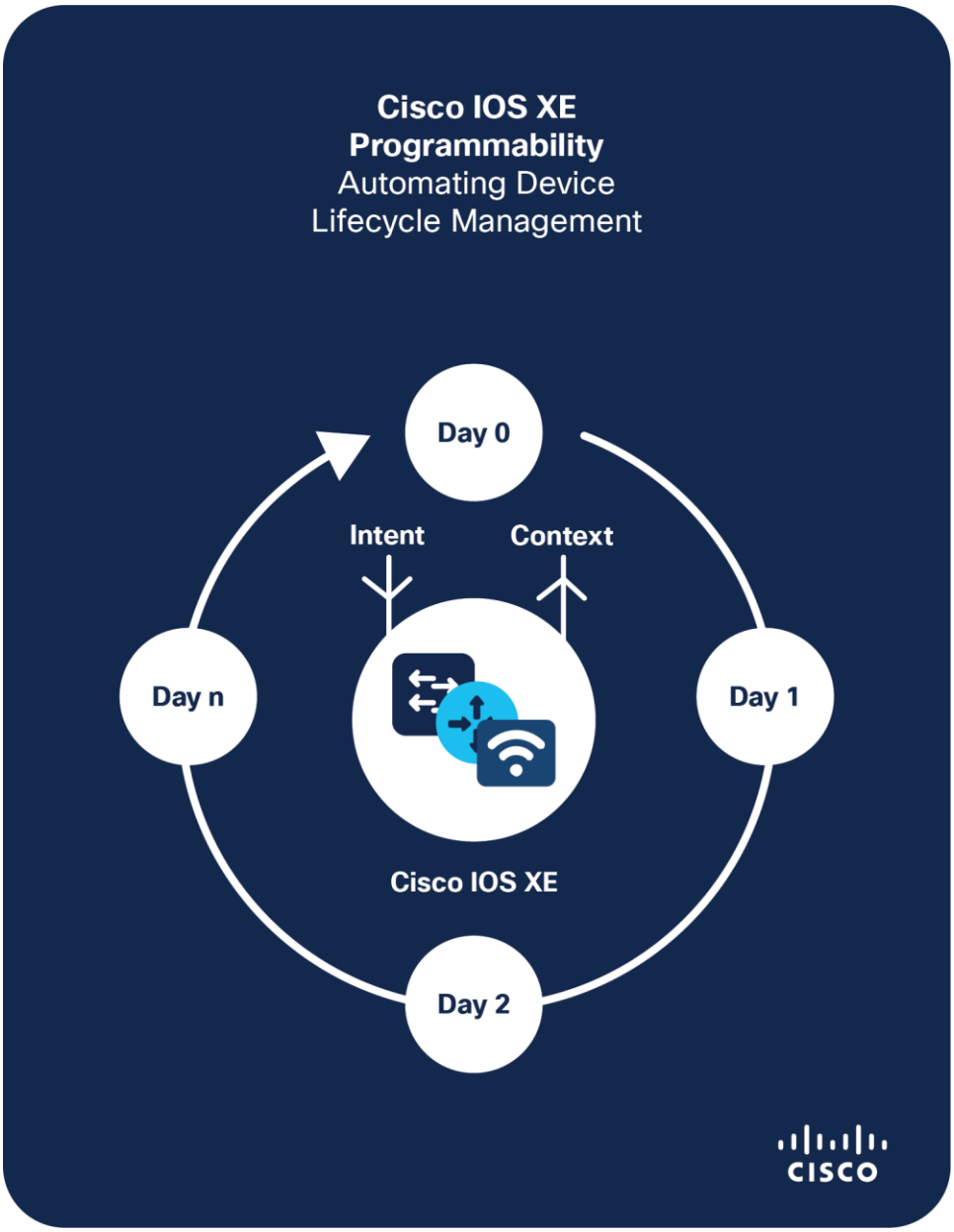


Figure 15.
Cisco IOS XE Programmability Book

Programmability Configuration Guide: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/176/b_176_programmability_cg.html?dtid=osscdc000283

Reference

YANG Suite: https://developer.cisco.com/yangsuite/

NCC: https://github.com/CiscoDevNet/ncc

Embedded Event Manager: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/eem/configuration/xe-17/eem-xe-17-book.html?dtid=osscdc000283

Questions?

Cisco DevNet: https://developer.cisco.com/

Cisco Communities: https://community.cisco.com

Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at **https://www.cisco.com/go/offices**.

Printed in USA C07-745008-00 11/21