gNMI Protocol

This feature describes the model-driven configuration and retrieval of operational data using the gNMI CAPABILITIES, GET and SET RPCs. gNMI version 0.4.0 is supported.

Restrictions for gNMI Protocol

- Use of the Origin field in the path message is not supported. A non-empty value will cause an error to be returned.
- Subscribe RPC services
- JSON, BYTES, PROTO, and ASCI encoding options

  JSON keys must contain a YANG-prefix where the namespace of the following elements differs from the parent. This means that the routed-vlan derived from augmentation in openconfig-vlan.yang must be entered as *oc-vlan:routed-vlan* because it is different from the namespace of the parent nodes (parent nodes have the prefix, oc-if)
- GetRequest:
  - Operational data type
  - Use models
- GetResponse Notifictions
  - Alias
  - Delete
- In a SetRequest, wildcards and all keys are not supported. Only fully-specified paths are supported.

Information About the gNMI Protocol

About GNMI

gNMI is gRPC network management protocol developed by Google. gNMI provides the mechanism to install, manipulate, and delete the configuration of network devices, and also 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.

When a failure occurs, the gNMI broker (GNMIB) will indicate an operational change of state from up to down, and all RPCs will return a service unavailable message until the database is up and running. Upon recovery, the GNMIB will indicate a change of operation state from down to up, and resume normal handling of RPCs.

Overview of RFC 7951

RFC 7951 defines JavaScript Object Notation (JSON) encoding for YANG data trees and their subtrees.

Instances of YANG data nodes (leafs, 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.

The "leaf" 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. The 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 (i.e., has no children, and an associated value) the value of that leaf is encoded directly - i.e., the "bare" value is specified (i.e., a JSON object is not required, and a bare JSON value is included).

The following example shows a leaf node definition:

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

The following is a valid JSON-encoded instance:

```
"foo": 123
```

gNMI GET Request

The gNMI GET RPC specifies how to retrieve one or more of the configuration attributes, state attributes, derived state attributes, or all attributes associated with a supported mode from a date tree. A GetRequest is sent from a client to the target to retrieve values from the data tree. A GetResponse is sent in response to a GetRequest.

The following example shows a Get Request on JSON structure:

```
Creating a path object for xpath: /oc-if:interfaces/interface[name=Loopback111]
++++++++ Sending get request: ++++++++
path {
  elem {
    name: "oc-if:interfaces"
  }
  elem {
    name: "interface"
    key {
      key: "name"
      value: "Loopback111"
    }
  }
}
encoding: JSON_IETF
++++++++ Recevied get response: ++++++++
notification {
  timestamp: 1521699434792345469
  update {
    path {
      elem {
        name: "oc-if:interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "\"Loopback111\""
        }
      }
    }
    val {
      json_ietf_val: "{\n\t\"openconfig-interfaces:name\":\t\"Loopback111\",\n\t\
       "openconfig-interfaces:config\":\t{\n\t\t\
       "openconfig-interfaces:type\":\t\"ianaift:softwareLoopback\",\n\t\t\
       "openconfig-interfaces:name\":\t\"Loopback111\",\n\t\t\
       "openconfig-interfaces:enabled\":\t\"true\"\n\t},\n\t\
       "openconfig-interfaces:state\":\t{\n\t\t\
       "openconfig-interfaces:type\":\t\"ianaift:softwareLoopback\",\n\t\t\
       "openconfig-interfaces:name\":\t\"Loopback111\",\n\t\t\
       "openconfig-interfaces:enabled\":\t\"true\",\n\t\t\
       "openconfig-interfaces:ifindex\":\t52,\n\t\t\
       "openconfig-interfaces:admin-status\":\t\"UP\",\n\t\t\
       "openconfig-interfaces:oper-status\":\t\"UP\",\n\t\t\
       "openconfig-interfaces:last-change\":\t2018,\n\t\t\
       "openconfig-interfaces:counters\":\t{\n\t\t\t\
       "openconfig-interfaces:in-octets\":\t0,\n\t\t\t\
       "openconfig-interfaces:in-unicast-pkts\":\t0,\n\t\t\t\
       "openconfig-interfaces:in-broadcast-pkts\":\t0,\n\t\t\t\
       "openconfig-interfaces:in-multicast-pkts\":\t0,\n\t\t\t\
       "openconfig-interfaces:in-discards\":\t0,\n\t\t\t\
       "openconfig-interfaces:in-errors\":\t0,\n\t\t\t\
       "openconfig-interfaces:in-unknown-protos\":\t0,\n\t\t\t\
       "openconfig-interfaces:out-octets\":\t0,\n\t\t\t\
       "openconfig-interfaces:out-unicast-pkts\":\t0,\n\t\t\t\
       "openconfig-interfaces:out-broadcast-pkts\":\t0,\n\t\t\t\
       "openconfig-interfaces:out-multicast-pkts\":\t0,\n\t\t\t\
       "openconfig-interfaces:out-discards\":\t0,\n\t\t\t\
       "openconfig-interfaces:out-errors\":\t0,\n\t\t\t\
       "openconfig-interfaces:last-clear\":\t2018\n\t\t},\n\t\t\
       "openconfig-platform:hardware-port\":\t\"Loopback111\"\n\t},\n\t\
       "openconfig-interfaces:subinterfaces\":\t{\n\t\t\
```

```
        "openconfig-interfaces:index\":\t0,\n\t\t\
        "openconfig-interfaces:config\":\t{\n\t\t\t\
        "openconfig-interfaces:index\":\t0,\n\t\t\t\
        "openconfig-interfaces:name\":\t\"Loopback111\",\n\t\t\t\
        "openconfig-interfaces:enabled\":\t\"true\"\n\t\t},\n\t\t\
        "openconfig-interfaces:state\":\t{\n\t\t\t\
        "openconfig-interfaces:index\":\t0,\n\t\t\t\
        "openconfig-interfaces:name\":\t\"Loopback111.0\",\n\t\t\t\
        "openconfig-interfaces:enabled\":\t\"true\",\n\t\t\t\
        "openconfig-interfaces:admin-status\":\t\"UP\",\n\t\t\t\
        "openconfig-interfaces:oper-status\":\t\"UP\",\n\t\t\t\
        "openconfig-interfaces:last-change\":\t2018,\n\t\t\t\
        "openconfig-interfaces:counters\":\t{\n\t\t\t\t\
        "openconfig-interfaces:in-octets\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:in-unicast-pkts\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:in-broadcast-pkts\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:in-multicast-pkts\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:in-discards\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:in-errors\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:out-octets\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:out-unicast-pkts\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:out-broadcast-pkts\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:out-multicast-pkts\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:out-discards\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:out-errors\":\t0,\n\t\t\t\t\
        "openconfig-interfaces:last-clear\":\t2018\n\t\t\t\t}\n\t\t},\n\t\t\
        "openconfig-if-ip:ipv6\":\t{\n\t\t\t\
        "openconfig-if-ip:config\":\t\"false\",\n\t\t\t\
        "openconfig-if-ip:state\":\t\"false\"\n\t\t}\n\t}\n}"
    }
  }
}
```

The following example shows a GetRequest on a leaf on JSON structure:

```
Creating a path object for xpath:
/oc-if:interfaces/interface[name=Loopback111]/state/oper-status
++++++++ Sending get request: ++++++++
path {
  elem {
    name: "oc-if:interfaces"
  }
  elem {
    name: "interface"
    key {
      key: "name"
      value: "Loopback111"
    }
  }
  elem {
    name: "state"
  }
  elem {
    name: "oper-status"
  }
}
encoding: JSON_IETF
++++++++ Recevied get response: ++++++++
notification {
  timestamp: 1521699326012374332
  update {
    path {
      elem {
```

```
        name: "oc-if:interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "\"Loopback111\""
        }
      }
      elem {
        name: "state"
      }
      elem {
        name: "oper-status"
      }
    }
    val {
      json_ietf_val: "\"UP\""
    }
  }
}
```

gNMI SetRequest

The Set RPC specifies how to set one or more configurable attributes associated with a supported model. A SetRequest is sent from a client to a target to update the values in the data tree.

In a SetRequest, only fully-specified (wildcards, and all keys-specified paths are not supported.) paths, and "json_ietf_val" or "json_val" TypedValue are supported. JSON keys must contain a YANG-prefix, in which the namespace of the following element differs from parent. The “routed-vlan” element derived from augmentation in openconfig-vlan.yang must be entered as “oc-vlan:routed-vlan”, because it is different from the namespace of the parent node (The parent node prefix is oc-if.).

The total set of deletes, replace, and updates contained in any one SetRequest is treated as a single transaction. If any subordinate element of the transaction fails; the entire transaction will be disallowed and rolled back. A SetResponse is sent back for a SetRequest.

The following example shows a SetRequest on JSON structure:

```
Creating UPDATE update for /oc-if:interfaces/interface[name=Loopback111]/config/
Creating a path object for xpath: /oc-if:interfaces/interface[name=Loopback111]/config/
++++++++ Sending set request: ++++++++
update {
  path {
    elem {
      name: "oc-if:interfaces"
    }
    elem {
      name: "interface"
      key {
        key: "name"
        value: "Loopback111"
      }
    }
    elem {
      name: "config"
    }
  }
  val {
    json_ietf_val: "{\"openconfig-interfaces:enabled\":\"false\"}"
```

```
  }
}
++++++++ Recevied set response: ++++++++
response {
  path {
    elem {
      name: "oc-if:interfaces"
    }
    elem {
      name: "interface"
      key {
        key: "name"
        value: "Loopback111"
      }
    }
    elem {
      name: "config"
    }
  }
  op: UPDATE
}
timestamp: 1521699342123890045
```

The following example shows a SetRequest on leaf on JSON structure:

```
Creating UPDATE update for /oc-if:interfaces/interface[name=Loopback111]/config/description
Creating a path object for xpath:
/oc-if:interfaces/interface[name=Loopback111]/config/description
++++++++ Sending set request: ++++++++
update {
  path {
    elem {
      name: "oc-if:interfaces"
    }
    elem {
      name: "interface"
      key {
        key: "name"
        value: "Loopback111"
      }
    }
    elem {
      name: "config"
    }
    elem {
      name: "description"
    }
  }
  val {
    json_ietf_val: "\"UPDATE DESCRIPTION\""
  }
}
++++++++ Recevied set response: ++++++++
response {
  path {
    elem {
      name: "oc-if:interfaces"
    }
    elem {
      name: "interface"
      key {
        key: "name"
```

```
        value: "Loopback111"
      }
    }
    elem {
      name: "config"
    }
    elem {
      name: "description"
    }
  }
  op: UPDATE
}
timestamp: 1521699342123890045
```

gNMI JSON_ietf_val

The JSON type indicates that the value is encoded as a JSON string as specified in RFC 7159. Additional types (such as, JSON_IETF) indicate specific additional characteristics of the encoding of the JSON data (particularly where they relate to serialisation of YANG-modeled data).

The following is a sample JSON_ietf_val message:

```
val {
  json_ietf_val:"{
   "oc-if:config": {
     "oc-if:description":
       "UPDATE DESCRIPTION"
   }
  }"
}
```

gNMI Error Messages

When errors occur, gNMI returns descriptive error messages. The following section displays some gNMI error messages.

The following sample error message is displayed when the path is invalid:

```
gNMI Error Response:
  <_Rendezvous of RPC that terminated with (StatusCode.TERMINATED,
   An error occurred while parsing provided xpath: unknown tag:
   "someinvalidxpath" Additional information: badly formatted or nonexistent path)>
```

The following sample error message is displayed for an unimplemented error:

```
gNMI Error Response:
  <_Rendezvous of RPC that terminated with (StatusCode.UNIMPLEMENTED,
    Requested encoding "ASCII" not supported)>
```

The following sample error message is displayed when the data element is empty:

```
gNMI Error Response:
  <_Rendezvous of RPC that terminated with (StatusCode.NOT_FOUND,
```

```
Empty set returned for path "/oc-if:interfaces/noinfohere")>
```

How to Enable the gNMI Protocol

Creating Certs with OpenSSL on Linux

Certs and trustpoint are only required for secure gNMI servers.

The following example shows how to create Certs with OpenSSL on a Linux machine:

```
# Setting up a CA
openssl genrsa -out rootCA.key 2048
openssl req -subj /C=/ST=/L=/O=/CN=rootCA -x509 -new -nodes -key rootCA.key -sha256 -out
rootCA.pem

# Setting up device cert and key
openssl genrsa -out device.key 2048
openssl req -subj /C=/ST=/L=/O=/CN=<hostnameFQDN> -new -key device.key -out device.csr
openssl x509 -req -in device.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out
device.crt -sha256
# Encrpyt device key - needed for input to IOS
openssl rsa -des3 -in device.key -out device.des3.key -passout pass:<password - remember
this for later>

# Setting up client cert and key
openssl genrsa -out client.key 2048
openssl req -subj /C=/ST=/L=/O=/CN=gnmi_client -new -key client.key -out client.csr
openssl x509 -req -in client.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out
client.crt -sha256
```

Installing Certs on a Device

The following example show how to install certs on a device:

```
# Send:
Device# configure terminal
Device(config)# crypto pki import trustpoint1 pem terminal password password1

# Receive:
% Enter PEM-formatted CA certificate.
% End with a blank line or "quit" on a line by itself.

# Send:
# Contents of rootCA.pem, followed by newline + 'quit' + newline:
-----BEGIN CERTIFICATE-----
<snip>
-----END CERTIFICATE-----
quit

# Receive:
% Enter PEM-formatted encrypted private General Purpose key.
% End with "quit" on a line by itself.

# Send:
# Contents of device.des3.key, followed by newline + 'quit' + newline:
-----BEGIN RSA PRIVATE KEY-----
```

```
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,D954FF9E43F1BA20
<snip>
-----END RSA PRIVATE KEY-----
quit

# Receive:
% Enter PEM-formatted General Purpose certificate.
% End with a blank line or "quit" on a line by itself.

# Send:
# Contents of device.crt, followed by newline + 'quit' + newline:
-----BEGIN CERTIFICATE-----
<snip>
-----END CERTIFICATE-----
quit

# Receive:
% PEM files import succeeded.
Device(config)#

# Send:
Device(config)# crypto pki trustpoint trustpoint1
Device(ca-trustpoint)# revocation-check none
Device(ca-trustpoint)# end
Device#
```

Enabling gNMI in Insecure Mode

Note This task is applicable in Cisco IOS XE Fuji 16.8.1 through Amsterdam 17.2.x.

In a Day Zero setup, first enable the device in insecure mode, then disable it, and enable the secure mode. To stop gNMI in insecure mode, use the **no gnmi-yang server** command.

Note gNMI insecure and secure servers can run simultaneously.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **gnmi-yang**
4. **gnmi-yang server**
5. **gnmi-yang port** *port-number*
6. **end**
7. **show gnmi-yang state**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**gnmi-yang** **Example:** `Device(config)# gnmi-yang`	Starts the gNMI process.
Step 4	**gnmi-yang server** **Example:** `Device(config)# gnmi-yang server`	Enables the gNMI server in insecure mode.
Step 5	**gnmi-yang port** *port-number* **Example:** `(Optional) Device(config)# gnmi-yang port 50000`	Sets the gNMI port to listen to. • The default insecure gNMI port is 9339.
Step 6	**end** **Example:** `Device(config)# end`	Exits global configuration mode and returns to privileged EXEC mode.
Step 7	**show gnmi-yang state** **Example:** `Device# show gnmi-yang state`	Displays the status of gNMI servers.

Example

The following is sample output from the **show gnmi-yang state** command:

```
Device# show gnmi-yang state

State Status
--------------------------------
Enabled Up
```

Enabling gNMI in Secure Mode

Note This task is applicable in Cisco IOS XE Fuji 16.8.1 through Amsterdam 17.2.x.

To stop gNMI in secure mode, use the **no gnmi-yang secure-server** command.

Note gNMI insecure and secure servers can run simultaneously.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **gnmi-yang**
4. **gnmi-yang secure-server**
5. **gnmi-yang secure-trustpoint** *trustpoint-name*
6. **gnmi-yang secure-client-auth**
7. **gnmi-yang secure-port**
8. **end**
9. **show gnmi-yang state**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**gnmi-yang** **Example:** `Device(config)# gnmi-yang`	Starts the gNMI process.
Step 4	**gnmi-yang secure-server** **Example:** `Device(config)# gnmi-yang secure-server`	Enables the gNMI server in secure mode.
Step 5	**gnmi-yang secure-trustpoint** *trustpoint-name* **Example:** `Device(config)# gnmi-yang secure-trustpoint trustpoint1`	Specifies the trustpoint and cert set that gNMI uses for authentication.

	Command or Action	Purpose
Step 6	**gnmi-yang secure-client-auth** **Example:** `Device(config)# gnmi-yang secure-client-auth`	(Optional) The gNMI process authenticates the client certificate against the root certificate.
Step 7	**gnmi-yang secure-port** **Example:** `Device(config)# gnmi-yang secure-port`	(Optional) Sets the gNMI port to listen to. • The default insecure gNMI port is 9339.
Step 8	**end** **Example:** `Device(config)# end`	Exits global configuration mode and returns to privileged EXEC mode.
Step 9	**show gnmi-yang state** **Example:** `Device# show gnmi-yang state`	Displays the status of gNMI servers.

Example

The following is sample output from the **show gnmi-yang state** command:

```
Device# show gnmi-yang state

State Status
--------------------------------
Enabled Up
```

Connecting the gNMI Client

The gNMI client is connected by using the client and root certificates that are previously configured.

The following example shows how to connect the gNMI client using Python:

```
# gRPC Must be compiled in local dir under path below:
>>> import sys
>>> sys.path.insert(0, "reference/rpc/gnmi/")
>>> import grpc
>>> import gnmi_pb2
>>> import gnmi_pb2_grpc
>>> gnmi_dir = '/path/to/where/openssl/creds/were/generated/'

# Certs must be read in as bytes
>>> with open(gnmi_dir + 'rootCA.pem', 'rb') as f:
>>>     ca_cert = f.read()
>>> with open(gnmi_dir + 'client.crt', 'rb') as f:
>>>     client_cert = f.read()
>>> with open(gnmi_dir + 'client.key', 'rb') as f:
>>>     client_key = f.read()

# Create credentials object
```

```
>>> credentials = grpc.ssl_channel_credentials(root_certificates=ca_cert,
private_key=client_key, certificate_chain=client_cert)

# Create a secure channel:
# Default port is 50052, can be changed on ios device with 'gnmi-yang secure-port ####'
>>> port = 50052
>>> host = <HOSTNAME FQDN>
>>> secure_channel = grpc.secure_channel("%s:%d" % (host, port), credentials)

# Create secure stub:
>>> secure_stub = gnmi_pb2_grpc.gNMIStub(secure_channel)

# Done! Let's test to make sure it works:
>>> secure_stub.Capabilities(gnmi_pb2.CapabilityRequest())
supported_models {
<snip>
}
supported_encodings: <snip>
gNMI_version: "0.4.0"
```

Configuration Examples for Enabling the gNMI Protocol

Example: Enabling the gNMI Protocol

Note This example is applicable in Cisco IOS XE Fuji 16.8.1 through Amsterdam 17.2.x.

Example: Enabling gNMI in Insecure Mode

The following example shows how to enable the gNMI server in insecure mode:

```
Device# configure terminal
Device(config)# gnmi-yang
Device(config)# gnmi-yang server
Device(config)# gnmi-yang port 50000 <The default port is 9339.>
Device(config)# end
Device#
```

Example: Enabling gNMI in Secure Mode

The following example shows how to enable the gNMI server in secure mode:

```
Device# configure terminal
Device(config)# gnmi-yang server
Device(config)# gnmi-yang secure-server
Device(config)# gnmi-yang secure-trustpoint trustpoint1
Device(config)# gnmi-yang secure-client-auth
Device(config)# gnmi-yang secure-port 50001 <The default port is 9339.>
Device(config)# end
Device#
```

Additional References for the gNMI Protocol

Related Documents

Related Topic	Document Title
Open config information	• https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi.proto
gNMI	https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-specification.md

Standards and RFCs

Standard/RFC	Title
RFC 7951	JSON Encoding of Data Modeled with YANG

MIBs

MIB	MIBs Link
CAPABILITIES-MIB	To locate and download MIBs for selected platforms, Cisco IOS releases, and feature sets, use Cisco MIB Locator found at the following URL: http://www.cisco.com/go/mibs

Technical Assistance

Description	Link
The Cisco Support website provides extensive online resources, including documentation and tools for troubleshooting and resolving technical issues with Cisco products and technologies. To receive security and technical information about your products, you can subscribe to various services, such as the Product Alert Tool (accessed from Field Notices), the Cisco Technical Services Newsletter, and Really Simple Syndication (RSS) Feeds. Access to most tools on the Cisco Support website requires a Cisco.com user ID and password.	http://www.cisco.com/support

Feature Information for the gNMI Protocol

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 1: Feature Information for the gNMI Protocol

Feature Name	Release	Feature Information
gNMI Protocol	Cisco IOS XE Fuji 16.8.1a	This feature describes the model-driven configuration and retrieval of operational data using the gNMI capabilities, GET and SET RPCs. This feature was implemented on the following platforms: • Cisco Catalyst 9300 Series Switches • Cisco Catalyst 9400 Series Switches • Cisco Catalyst 9500 Series Switches
	Cisco IOS XE Gibraltar 16.10.1	gNMI namespaces and gNMI wildcards support were added to the following platforms: • Cisco Catalyst 9300 Series Switches • Cisco Catalyst 9400 Series Switches • Cisco Catalyst 9500 Series Switches