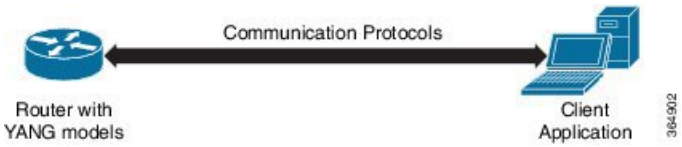
# Components to Use Data Models

The process of automating configurations in a network involves the use of these core components:

- **Client application:** manages and monitors the configuration of the devices in the network.
- **Router:** acts as a server, responds to requests from the client application and configures the devices in the network.
- **YANG module:** describes configuration and operational data of the router, and perform actions.
- **Communication protocol:** provides mechanisms to install, manipulate, and delete the configuration of network devices.

Figure 2 shows the interplay of the core components.

***Figure 1: Components in Using Data Models***




This chapter describes these two components:


- YANG Module, on page 1
- Communication Protocols, on page 5
- YANG Actions, on page 15


# YANG Module

A YANG module defines a data model through the data of the router, and the hierarchical organization and constraints on that data. Each module is uniquely identified by a namespace URL. The YANG models describe the configuration and operational data, perform actions, remote procedure calls, and notifications for network devices.

The YANG models must be obtained from the router. The models define a valid structure for the data that is exchanged between the router and the client. The models are used by NETCONF and gRPC-enabled applications.

YANG models can be:

- **Cisco-specific models:** For a list of supported models and their representation, see Github.
- **Common models:** These models are industry-wide standard YANG models from standard bodies, such as IETF and IEEE. These models are also called Open Config (OC) models. Like synthesized models, the OC models have separate YANG models defined for configuration data and operational data, and actions.

For a list of supported OC models and their representation, see Github.

For more details about YANG, refer RFC 6020 and 6087.

# Components of a YANG Module

A YANG module defines a single data model. However, a module can reference definitions in other modules and sub-modules by using one of these statements:

- **import** imports external modules
- **include** includes one or more sub-modules
- **augment** provides augmentations to another module, and defines the placement of new nodes in the data model hierarchy
- **when** defines conditions under which new nodes are valid
- **prefix** references definitions in an imported module

The YANG models configure a feature, retrieve the operational state of the router, and perform actions.

**Note** The gRPC YANG path or JSON data is based on YANG module name and not YANG namespace.

### Example: Configuration YANG Model for AAA

The YANG models used to configure a feature is denoted by -cfg.

```
(snippet)
module Cisco-IOS-XR-aaa-locald-cfg {

  /*** NAMESPACE / PREFIX DEFINITION ***/

  namespace "http://cisco.com/ns/yang/Cisco-IOS-XR-aaa-locald-cfg";


  prefix "aaa-locald-cfg";

  /*** LINKAGE (IMPORTS / INCLUDES) ***/

  import Cisco-IOS-XR-types { prefix "xr"; }

  import Cisco-IOS-XR-aaa-lib-cfg { prefix "a1"; }

  /*** META INFORMATION ***/

  organization "Cisco Systems, Inc.";
          ......................
          ...................... (truncated)
```

### Example: Operational YANG Model for AAA

The YANG models used to retrieve operational data is denoted by -oper.

```
 (snippet)
module Cisco-IOS-XR-aaa-locald-oper {

  /*** NAMESPACE / PREFIX DEFINITION ***/

  namespace "http://cisco.com/ns/yang/Cisco-IOS-XR-aaa-locald-oper";


  prefix "aaa-locald-oper";

  /*** LINKAGE (IMPORTS / INCLUDES) ***/

  import Cisco-IOS-XR-types { prefix "xr"; }

  include Cisco-IOS-XR-aaa-locald-oper-sub1 {
    revision-date 2015-01-07;
  }

  /*** META INFORMATION ***/

  organization "Cisco Systems, Inc.";
    ......................
    ...................... (truncated)
```

**Note** A module can include any number of sub-modules; each sub-module belongs to only one module. The names of all standard modules and sub-modules must be unique.

### Example: NETCONF Action for OSPFv3

The YANG models used to perform actions is denoted by -act.

```
 (snippet)
clear ospfv3 1 vrf vrf1 statistics neighbor 2.2.2.2
RPC message based on the new ospfv3 yang model-
 <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <act-ospfv3-instance-vrf xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv6-ospfv3-act">

     <instance>
       <instance-identifier>1</instance-identifier>
        <vrf>
         <vrf-name>vrf1</vrf-name>
         <stats>
          <neighbor>
           <neighbor-id>2.2.2.2</neighbor-id>
          </neighbor>
         </stats>
        </vrf>
     </instance>
   </act-ospfv3-instance-vrf>
 </rpc>
```

# Structure of YANG Models

YANG data models can be represented in a hierarchical, tree-based structure with nodes. This representation makes the models easy to understand.

Each feature has a defined YANG model, which is synthesized from schemas. A model in a tree format includes:

- Top level nodes and their subtrees
- Subtrees that augment nodes in other YANG models
- Custom RPCs

YANG defines four node types. Each node has a name. Depending on the node type, the node either defines a value or contains a set of child nodes. The nodes types for data modeling are:

- leaf node - contains a single value of a specific type
- leaf-list node - contains a sequence of leaf nodes
- list node - contains a sequence of leaf-list entries, each of which is uniquely identified by one or more key leaves
- container node - contains a grouping of related nodes that have only child nodes, which can be any of the four node types

### Example: Structure of CDP Data Model

Cisco Discovery Protocol (CDP) configuration has an inherent augmented model (interface-configuration). The augmentation indicates that CDP can be configured at both the global configuration level and the interface configuration level. The data model for CDP interface manager in tree structure is:

```
module: Cisco-IOS-XR-cdp-cfg
   +--rw cdp
      +--rw timer?               uint32
      +--rw advertise-v1-only?   empty
      +--rw enable?              boolean
      +--rw hold-time?           uint32
      +--rw log-adjacency?       empty
augment /a1:interface-configurations/a1:interface-configuration:
   +--rw cdp
      +--rw enable?   empty
```

In the CDP YANG model, the augmentation is expressed as:

```
augment "/a1:interface-configurations/a1:interface-configuration" {
    container cdp {
      description "Interface specific CDP configuration";
      leaf enable {
        type empty;
        description "Enable or disable CDP on an interface";
      }
    }
    description
      "This augment extends the configuration data of
      'Cisco-IOS-XR-ifmgr-cfg'";
  }
```

**CDP Operational YANG:**

```
module: Cisco-IOS-XR-cdp-oper
   +--ro cdp
      +--ro nodes
         +--ro node* [node-name]
            +--ro neighbors
            |  +--ro details
            |  |  +--ro detail*
            |  |     +--ro interface-name?   xr:Interface-name
            |  |     +--ro device-id?        string
            |  |     +--ro cdp-neighbor*
            |  |        +--ro detail
            |  |        |  +--ro network-addresses
            |  |        |  |  +--ro cdp-addr-entry*
            |  |        |  |     +--ro address
            |  |        |  |        +--ro address-type?   Cdp-l3-addr-protocol
            |  |        |  |        +--ro ipv4-address?   inet:ipv4-address
            |  |        |  |        +--ro ipv6-address?   In6-addr
            |  |        |  +--ro protocol-hello-list
            |  |        |  |  +--ro cdp-prot-hello-entry*
            |  |        |  |     +--ro hello-message?   yang:hex-string
            |  |        |  +--ro version?               string
            |  |        |  +--ro vtp-domain?            string
            |  |        |  +--ro native-vlan?           uint32
            |  |        |  +--ro duplex?                Cdp-duplex
            |  |        |  +--ro system-name?           string
            |  |        +--ro receiving-interface-name?   xr:Interface-name
            |  |        +--ro device-id?                  string
            |  |        +--ro port-id?                    string
            |  |        +--ro header-version?             uint8
            |  |        +--ro hold-time?                  uint16
            |  |        +--ro capabilities?               string
            |  |        +--ro platform?                   string

.............................................. (truncated)
```

# Communication Protocols

Communication protocols establish connections between the router and the client. The protocols help the client to consume the YANG data models to, in turn, automate and programme network operations.

YANG uses one of these protocols :

- Network Configuration Protocol (NETCONF)
- gRPC (google-defined Remote Procedure Calls)

The transport and encoding mechanisms for these two protocols are shown in the table:

| Protocol | Transport | Encoding/ Decoding |
|---|---|---|
| NETCONF | ssh | xml |
| gRPC | http/2 | json |

# NETCONF Protocol

NETCONF provides mechanisms to install, manipulate, or delete the configuration of network devices. It uses an Extensible Markup Language (XML)-based data encoding for the configuration data, as well as protocol messages. Use **ssh server capability netconf-xml** command to enable NETCONF to reach XML subsystem via port 22. NETCONF uses a simple RPC-based (Remote Procedure Call) mechanism to facilitate communication between a client and a server. The client can be a script or application that runs as part of a network manager. The server is a network device such as a router.

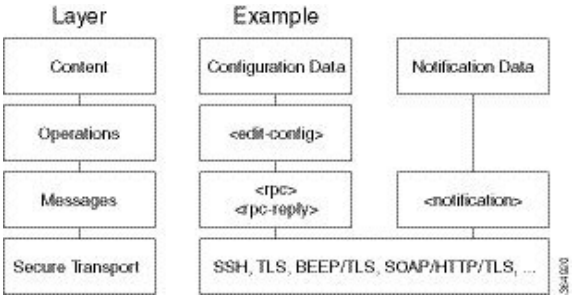
### NETCONF Session

A NETCONF session is the logical connection between a network configuration application (client) and a network device (router). The configuration attributes can be changed during any authorized session; the effects are visible in all sessions. NETCONF is connection-oriented, with SSH as the underlying transport. NETCONF sessions are established with a "hello" message, where features and capabilities are announced. Sessions are terminated using *close* or *kill* messages.

### NETCONF Layers

NETCONF can be partitioned into four layers:

***Figure 2: NETCONF Layers***




- **Content layer:** includes configuration and notification data
- **Operations layer:** defines a set of base protocol operations invoked as RPC methods with XML-encoded parameters
- **Messages layer:** provides a simple, transport-independent framing mechanism for encoding RPCs and notifications
- **Secure Transport layer:** provides a communication path between the client and the server

For more information about NETCONF, refer RFC 6241.

## NETCONF Operations

NETCONF defines one or more configuration datastores and allows configuration operations on the datastores. A configuration datastore is a complete set of configuration data that is required to get a device from its initial default state into a desired operational state. The configuration datastore does not include state data or executive commands.

The base protocol includes the following NETCONF operations:

```
|  +--Get-config
|  +--Edit-Config
|     +--Merge
|     +--Replace
|     +--Create
|     +--Delete
|     +--Remove
|     +--Default-Operations
|        +--Merge
|        +--Replace
|        +--None
|  +--Get
|  +--Lock
|  +--UnLock
|  +--Close-Session
|  +--Kill-Session
```

| NETCONF Operation | Description | Example |
|---|---|---|
| <get-config> | Retrieves all or part of a specified configuration from a named data store | Retrieve specific interface configuration details from running configuration using filter option<br><br>`<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><get-config><source><running/></source><filter><interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg"\><interface-configuration><active>act</active><interface-name>TenGigE0/0/0/2/0</interface-name></interface-configuration></interface-configurations></filter></get-config></rpc>` |
| <get> | Retrieves running configuration and device state information | Retrieve all acl configuration and device state information.<br><br>`Request:<get><filter><ipv4-acl-and-prefix-list xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-acl-oper"/></filter></get>` |

| NETCONF Operation | Description | Example |
|---|---|---|
| <edit-config> | Loads all or part of a specified configuration to the specified target configuration | Configure ACL configs using **Merge** operation<br><br>`<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <edit-config> <target><candidate/></target> <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0"> <ipv4-acl-and-prefix-list xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-acl-cfg" xc:operation=”merge”> <accesses> <access> <access-list-name>aclv4-1</access-list-name> <access-list-entries> <access-list-entry> <sequence-number>10</sequence-number> <remark>GUEST</remark> </access-list-entry> <access-list-entry> <sequence-number>20</sequence-number> <grant>permit</grant> <source-network> <source-address>172.0.0.0</source-address> <source-wild-card-bits>0.0.255.255</source-wild-card-bits> </source-network> </access-list-entry> </access-list-entries> </access> </accesses> </ipv4-acl-and-prefix-list> </config> </edit-config> </rpc>`<br><br>`Commit:`<br>`<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <commit/> </rpc>` |
| <lock> | Allows the client to lock the entire configuration datastore system of a device | Lock the running configuration.<br>`Request:`<br>`<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <lock> <target> <running/> </target> </lock> </rpc>`<br><br>`Response :`<br>`<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`<br><br>`<ok/> </rpc-reply>` |

| NETCONF Operation | Description | Example |
|---|---|---|
| <Unlock> | Releases a previously locked configuration.<br>An <unlock> operation will not succeed if either of the following conditions is true:<br>• The specified lock is not currently active.<br>• The session issuing the <unlock> operation is not the same session that obtained the lock. | Lock and unlock the running configuration from the same session.<br>`Request:`<br>`rpc message-id="101"`<br>`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`<br>`<unlock>`<br>`<target>`<br>`<running/>`<br>`</target>`<br>`</unlock>`<br>`</rpc>`<br><br>`Response -`<br>`<rpc-reply message-id="101"`<br>`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`<br>`<ok/>`<br>`</rpc-reply>` |
| <close-session> | Closes the session. The server releases any locks and resources associated with the session and closes any associated connections. | Close a NETCONF session.<br>`Request :`<br>`<rpc message-id="101"`<br>`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`<br>`<close-session/>`<br>`</rpc>`<br><br>`Response:`<br>`<rpc-reply message-id="101"`<br>`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`<br>`<ok/>`<br>`</rpc-reply>` |
| <kill-session> | Terminates operations currently in process, releases locks and resources associated with the session, and close any associated connections. | Terminate a session if the ID is other session ID.<br>`Request:`<br>`<rpc message-id="101"`<br>`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`<br>`<kill-session>`<br>`<session-id>4</session-id>`<br>`</kill-session>`<br>`</rpc>`<br><br>`Response:`<br>`<rpc-reply message-id="101"`<br>`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`<br>`<ok/>`<br>`</rpc-reply>` |

### Example: NETCONF Operation to Get Configuration

This example shows how a NETCONF <get-config> request works for CDP feature.

The client initiates a message to get the current configuration of CDP running on the router. The router responds with the current CDP configuration.

| Netconf Request (Client to Router) | Netconf Response (Router to Client) |
|---|---|
| <rpc message-id="101"<br>xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><br><br><get-config><br><source><running/></source><br><filter><br><cdp<br>xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-cdp-cfg"/><br></filter><br></get-config><br></rpc> | <?xml version="1.0"?><br><rpc-reply message-id="101"<br>xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><br><br><data><br><cdp<br>xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-cdp-cfg"><br><br><timer>10</timer><br><enable>true</enable><br><log-adjacency></log-adjacency><br><hold-time>200</hold-time><br><advertise-v1-only></advertise-v1-only><br></cdp><br>#22<br></data><br></rpc-reply> |

The <rpc> element in the request and response messages enclose a NETCONF request sent between the client and the router. The `message-id` attribute in the <rpc> element is mandatory. This attribute is a string chosen by the sender and encodes an integer. The receiver of the <rpc> element does not decode or interpret this string but simply saves it to be used in the <rpc-reply> message. The sender must ensure that the `message-id` value is normalized. When the client receives information from the server, the <rpc-reply> message contains the same `message-id`.

# gRPC Protocol

gRPC is an open-source RPC framework. It is based on Protocol Buffers (Protobuf), which is an open source binary serialization protocol. gRPC provides a flexible, efficient, automated mechanism for serializing structured data, like XML, but is smaller and simpler to use. The user needs to define the structure by defining protocol buffer message types in .proto files. Each protocol buffer message is a small logical record of information, containing a series of name-value pairs.

gRPC encodes requests and responses in binary. gRPC is extensible to other content types along with Protobuf. The Protobuf binary data object in gRPC is transported over HTTP/2.

**Note** It is recommended to configure TLS before enabling gRPC. Enabling gRPC protocol uses the default HTTP/2 transport with no TLS enabled on TCP. gRPC mandates AAA authentication and authorization for all gRPC requests. If TLS is not configured, the authentication credentials are transferred over the network unencrypted. Non-TLS mode can only be used in secure internal network.

gRPC supports distributed applications and services between a client and server. gRPC provides the infrastructure to build a device management service to exchange configuration and operational data between a client and a server. The structure of the data is defined by YANG models.

Cisco gRPC IDL uses the protocol buffers interface definition language (IDL) to define service methods, and define parameters and return types as protocol buffer message types. The gRPC requests are encoded and sent to the router using JSON. Clients can invoke the RPC calls defined in the IDL to program the router.

The following example shows the syntax of the proto file for a gRPC configuration:

```
syntax = "proto3";

package IOSXRExtensibleManagabilityService;

service gRPCConfigOper {

    rpc GetConfig(ConfigGetArgs) returns(stream ConfigGetReply) {};

    rpc MergeConfig(ConfigArgs) returns(ConfigReply) {};

    rpc DeleteConfig(ConfigArgs) returns(ConfigReply) {};

    rpc ReplaceConfig(ConfigArgs) returns(ConfigReply) {};

    rpc CliConfig(CliConfigArgs) returns(CliConfigReply) {};

    rpc GetOper(GetOperArgs) returns(stream GetOperReply) {};

    rpc CommitReplace(CommitReplaceArgs) returns(CommitReplaceReply) {};
}
message ConfigGetArgs {
     int64 ReqId = 1;
     string yangpathjson = 2;
}

message ConfigGetReply {
    int64 ResReqId = 1;
    string yangjson = 2;
    string errors = 3;
}

message GetOperArgs {
     int64 ReqId = 1;
     string yangpathjson = 2;
}

message GetOperReply {
    int64 ResReqId = 1;
    string yangjson = 2;
    string errors = 3;
}

message ConfigArgs {
    int64 ReqId = 1;
    string yangjson = 2;

}

message ConfigReply {
    int64 ResReqId = 1;
    string errors = 2;
}

message CliConfigArgs {
    int64 ReqId = 1;
    string cli = 2;
}

message CliConfigReply {
    int64 ResReqId = 1;
    string errors = 2;
}

message CommitReplaceArgs {
```

```
     int64 ReqId = 1;
     string cli = 2;
     string yangjson = 3;
}

message CommitReplaceReply {
     int64 ResReqId = 1;
     string errors = 2;
}
```

Example for gRPCExec configuration:

```
service gRPCExec {
    rpc ShowCmdTextOutput(ShowCmdArgs) returns(stream ShowCmdTextReply) {};
    rpc ShowCmdJSONOutput(ShowCmdArgs) returns(stream ShowCmdJSONReply) {};

}

message ShowCmdArgs {
     int64 ReqId = 1;
     string cli = 2;
}

message ShowCmdTextReply {
    int64 ResReqId =1;
    string output = 2;
    string errors = 3;
}
```

Example for OpenConfiggRPC configuration:

```
service OpenConfiggRPC {
     rpc SubscribeTelemetry(SubscribeRequest) returns (stream SubscribeResponse) {};
     rpc UnSubscribeTelemetry(CancelSubscribeReq) returns (SubscribeResponse) {};
     rpc GetModels(GetModelsInput) returns (GetModelsOutput) {};
}

message GetModelsInput {
     uint64  requestId   = 1;
     string  name        = 2;
    string  namespace   = 3;
    string  version     = 4;
    enum MODLE_REQUEST_TYPE {
         SUMMARY = 0;
         DETAIL  = 1;
     }
     MODLE_REQUEST_TYPE requestType = 5;
}

message GetModelsOutput {
     uint64  requestId   = 1;
     message ModelInfo {
         string  name        = 1;
         string  namespace   = 2;
        string  version     = 3;
         GET_MODEL_TYPE  modelType = 4;
         string modelData = 5;
     }
     repeated ModelInfo models = 2;
     OC_RPC_RESPONSE_TYPE responseCode = 3;
```

```
     string msg = 4;
}
```

# gRPC Operations

The gRPC operations include:

| gRPC Operation | Description |
|---|---|
| GetConfig | Retrieves a configuration |
| GetModels | Gets the supported Yang models on the router |
| MergeConfig | Appends to an existing configuration |
| DeleteConfig | Deletes a configuration |
| ReplaceConfig | Modifies a part of an existing configuration |
| CommitReplace | Replaces existing configuration with the new configuration file provided |
| GetOper | Gets operational data using JSON |
| CliConfig | Invokes the CLI configuration |
| ShowCmdTextOutput | Displays the output of show command |
| ShowCmdJSONOutput | Displays the JSON output of show command |

### Example: Get Configuration for a Specific Interface

This example shows getting configuration for a specific interface using gRPC GetConfig operation.

```
{
    "Cisco-IOS-XR-ifmgr-cfg:interface-configurations": {
        "interface-configuration": [
            {
                "active": "act",
                "interface-name": "HundredGigE0/3/0/0"
            }
        ]
    }
}
```

### Example: Delete Configuration for CDP Container

This example shows how a gRPC DeleteConfig operation deletes a CDP container and a leaf within the container. The DeleteConfig argument identifies the resource using the YANG node. The value of the YANG node is ignored and set to null.

In this example, a CDP container is deleted:

```
{
```

```
"Cisco-IOS-XR-cdp-cfg:cdp": [null]
}
```

In this example, a leaf value for `hold-time` in the CDP container is deleted:

```
{
"Cisco-IOS-XR-cdp-cfg:cdp":
{
"hold-time": [null]
}
}
```

#### Example: Merge Configuration for CDP Timer

This example shows merging configuration for CDP timer using gRPC MergeConfig operation.

```
{
    "Cisco-IOS-XR-cdp-cfg:cdp": {
        "timer": 50
    }
}
```

#### Example: Get Operational Data for Interfaces

This example getting operational data for interfaces using gRPC GetOper operation.

```
{
    "Cisco-IOS-XR-ifmgr-oper:interface-properties": [null]
}
```

## gRPC Network Management Interface

gRPC Network Management Interface (gNMI) is a gRPC-based network management protocol used to modify, install or delete configuration from network devices. It is also used to view operational data, control and generate telemetry streams from a target device to a data collection system. It uses a single protocol to manage configurations and stream telemetry data from network devices.

gNNMI uses gRPC as the transport protocol and the configuration is same as that of gRPC. These gNMI RPCs are supported:

| gNMI RPC | gNMI RPC Request | Description |
|---|---|---|
| Capabilities | | Initial handshake between the network device (server) and the client to exchange capability information such as supported data models |
| Set | `message SetRequest {}` | Modifies data associated with a model on a network device from a client |

| gNMI RPC | gNMI RPC Request | Description |
|---|---|---|
| Get | `message GetRequest {}` | Retrieves data from a network device |
| Subscribe | `message SubscribeRequest {}` | Control data subscriptions on server |

For more information about gNMI, see Github repository.

# YANG Actions

IOS XR actions are RPC statements that trigger an operation or execute a command on the router. Theses actions are defined as YANG models using RPC statements. An action is executed when the router receives the corresponding NETCONF RPC request. Once the router executes an action, it replies with a NETCONF RPC response.

For example, **ping** command is a supported action. That means, a YANG model is defined for the **ping** command using RPC statements. This command can be executed on the router by initiating the corresponding NETCONF RPC request.

**Note** NETCONF supports XML format, and gRPC supports JSON format.

For the list of supported actions, see the following table:

| Actions | YANG Models |
|---|---|
| logmsg | Cisco-IOS-XR-syslog-act |
| snmp | Cisco-IOS-XR-snmp-test-trap-act |
| rollback | Cisco-IOS-XR-cfgmgr-rollback-act |
| clear isis | Cisco-IOS-XR-isis-act |
| clear bgp | Cisco-IOS-XR-ipv4-bgp-act |

### Example: PING NETCONF Action

This use case shows the IOS XR NETCONF action request to run the ping command on the router.

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <ping xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ping-act">
  <destination>
   <destination>1.2.3.4</destination>
  </destination>
 </ping>
</rpc>
```

This section shows the NETCONF action response from the router.

```
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
```

```
 <ping-response xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ping-act">
  <ipv4>
   <destination>1.2.3.4</destination>
   <repeat-count>5</repeat-count>
   <data-size>100</data-size>
   <timeout>2</timeout>
   <pattern>0xabcd</pattern>
   <rotate-pattern>0</rotate-pattern>
   <reply-list>
    <result>!</result>
    <result>!</result>
    <result>!</result>
    <result>!</result>
    <result>!</result>
   </reply-list>
   <hits>5</hits>
   <total>5</total>
   <success-rate>100</success-rate>
   <rtt-min>1</rtt-min>
   <rtt-avg>1</rtt-avg>
   <rtt-max>1</rtt-max>
  </ipv4>
 </ping-response>
</rpc-reply>
```

### Example: XR Process Restart Action

This example shows the process restart action sent to NETCONF agent.

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <sysmgr-process-restart xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-sysmgr-act">
     <process-name>processmgr</process-name>
     <location>0/RP0/CPU0</location>
  </sysmgr-process-restart>
</rpc>
```

This example shows the action response received from the NETCONF agent.

```
<?xml version="1.0"?>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <ok/>
</rpc-reply>
```

### Example: Copy Action

This example shows the RPC request and response for `copy` action:

**RPC request:**

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <copy xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-copy-act">
    <sourcename>//root:<location>/100MB.txt</sourcename>
    <destinationname>/</destinationname>
    <sourcefilesystem>ftp:</sourcefilesystem>
    <destinationfilesystem>harddisk:</destinationfilesystem>
    <destinationlocation>0/RSP1/CPU0</destinationlocation>
  </copy>
</rpc>
```

**RPC response:**

```
<?xml version="1.0"?>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <response xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-copy-act">Successfully
completed copy operation</response>
</rpc-reply>

8.261830565s elapsed
```

### Example: Delete Action

This example shows the RPC request and response for `delete` action:

**RPC request:**

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
<delete xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-delete-act">
    <name>harddisk:/netconf.txt</name>
  </delete>
</rpc>
```

**RPC response:**

```
<?xml version="1.0"?>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <response xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-delete-act">Successfully
 completed delete operation</response>
</rpc-reply>

395.099948ms elapsed
```