Drive Network Automation Using Programmable YANG Data Models

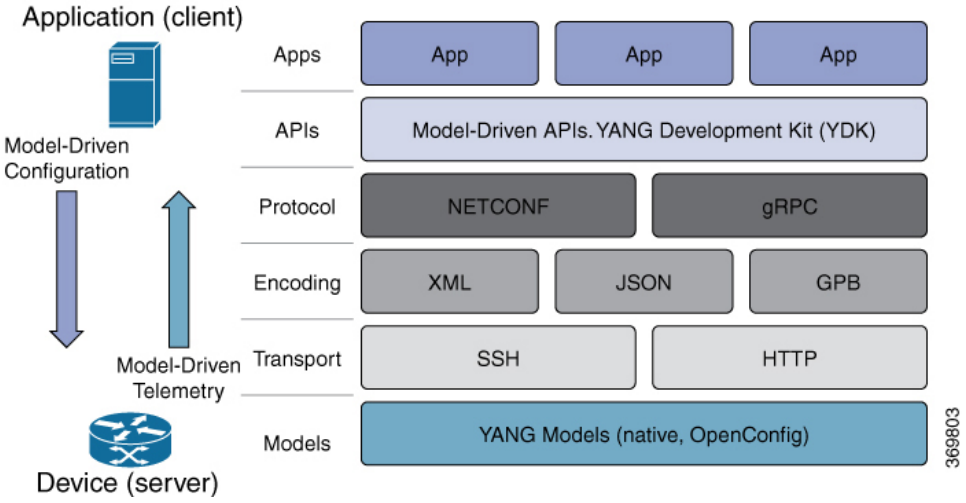
Typically, a network operation center is a heterogeneous mix of various devices at multiple layers of the network. Such network centers require bulk automated configurations to be accomplished seamlessly. CLIs are widely used for configuring and extracting the operational details of a router. But the general mechanism of CLI scraping is not flexible and optimal. Small changes in the configuration require rewriting scripts multiple times. Bulk configuration changes through CLIs are cumbersome and error-prone. These limitations restrict automation and scale. To overcome these limitations, you need an automated mechanism to manage your network.

Cisco IOS XR supports a programmatic way of configuring and collecting operational data of a network device using data models. They replace the process of manual configuration, which is proprietary, and highly text-based. The data models are written in an industry-defined language and is used to automate configuration task and retrieve operational data across heterogeneous devices in a network. Although configurations using CLIs are easier and human-readable, automating the configuration using model-driven programmability results in scalability.

Model-driven programmability provides a simple, flexible and rich framework for device programmability. This programmability framework provides multiple choices to interface with an IOS XR device in terms of transport, protocol and encoding. These choices are decoupled from the models for greater flexibility.

The following image shows the layers in model-driven programmability:

Figure 1: Model-driven Programmability Layers



Data models provides access to the capabilities of the devices in a network using Network Configuration Protocol (NETCONF Protocol) or google-defined Remote Procedure Calls (gRPC Protocol). The operations on the router are carried out by the protocols using YANG models to automate and programme operations in a network.

Benefits of Data Models

Configuring routers using data models overcomes drawbacks posed by traditional router management because the data models:

- Provide a common model for configuration and operational state data, and perform NETCONF actions.
- Use protocols to communicate with the routers to get, manipulate and delete configurations in a network.
- Automate configuration and operation of multiple routers across the network.

This article describes how you benefit from using data models to programmatically manage your network operations.

- YANG Data Model, on page 2
- Access the Data Models, on page 8
- Prevent Partial Pseudo-Atomic Committed Configurations, on page 10
- Communication Protocols, on page 11
- YANG Actions, on page 12

YANG Data Model

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.

Note gRPC is supported only in 64-bit platforms.

- **Cisco-specific models:** For a list of supported models and their representation, see Native models.
- **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.

YANG models can be: For a list of supported OC models and their representation, see OC models.

All data models are stamped with semantic version `1.0.0` as baseline from release 7.0.1 and later.

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

Data models handle the following types of requirements on routers (RFC 6244):

- **Configuration data:** A set of writable data that is required to transform a system from an initial default state into its current state. For example, configuring entries of the IP routing tables, configuring the interface MTU to use a specific value, configuring an ethernet interface to run at a given speed, and so on.
- **Operational state data:** A set of data that is obtained by the system at runtime and influences the behavior of the system in a manner similar to configuration data. However, in contrast to configuration data, operational state data is transient. The data is modified by interactions with internal components or other systems using specialized protocols. For example, entries obtained from routing protocols such as OSPF, attributes of the network interfaces, and so on.
- **Actions:** A set of NETCONF actions that support robust network-wide configuration transactions. When a change is attempted that affects multiple devices, the NETCONF actions simplify the management of failure scenarios, resulting in the ability to have transactions that will dependably succeed or fail atomically.

For more information about Data Models, see RFC 6244.

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

Structure of LLDP Data Model

The Link Layer Discovery Protocol (LLDP) data model is represented in the following structure:

```
$ cat Cisco-IOS-XR-ethernet-lldp-cfg.yang
module Cisco-IOS-XR-ethernet-lldp-cfg {

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

  namespace "http://cisco.com/ns"+
    "/yang/Cisco-IOS-XR-ethernet-lldp-cfg";


  prefix "ethernet-lldp-cfg";

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

  import cisco-semver { prefix "semver"; }

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

  /*** META INFORMATION ***/

  organization "Cisco Systems, Inc.";

  contact
    "Cisco Systems, Inc.
     Customer Service

     Postal: 170 West Tasman Drive
     San Jose, CA 95134

     Tel: +1 800 553-NETS

     E-mail: cs-yang@cisco.com";

  description
    "This module contains a collection of YANG definitions
     for Cisco IOS-XR ethernet-lldp package configuration.

     This module contains definitions
     for the following management objects:
       lldp: Enable LLDP, or configure global LLDP subcommands

     This YANG module augments the
       Cisco-IOS-XR-ifmgr-cfg
     module with configuration data.

     Copyright (c) 2013-2019 by Cisco Systems, Inc.
     All rights reserved.";

  revision "2019-04-05" {
    description
      "Establish semantic version baseline.";
    semver:module-version "1.0.0";
  }
```

```
  revision "2017-05-01" {
    description
      "Fixing backward compatibility error in module.";
  }

  revision "2015-11-09" {
    description
      "IOS XR 6.0 revision.";
  }


  container lldp {
    description "Enable LLDP, or configure global LLDP subcommands";

    container tlv-select {
      presence "Indicates a tlv-select node is configured.";
      description "Selection of LLDP TLVs to disable";

      container system-name {
        description "System Name TLV";
        leaf disable {
          type boolean;
          default "false";
          description "disable System Name TLV";
        }
      }

      container port-description {
        description "Port Description TLV";
        leaf disable {
          type boolean;
          default "false";
          description "disable Port Description TLV";
        }
      }
..... (snipped) .....
      container management-address {
        description "Management Address TLV";
        leaf disable {
          type boolean;
          default "false";
          description "disable Management Address TLV";
        }
      }
      leaf tlv-select-enter {
        type boolean;
        mandatory true;
        description "enter lldp tlv-select submode";
      }
    }
    leaf holdtime {
      type uint32 {
        range "0..65535";
      }
      description
        "Length  of time  (in sec) that receiver must
        keep this packet";

  ..... (snipped) .....
  }

  augment "/a1:interface-configurations/a1:interface-configuration" {
```

```
    container lldp {
      presence "Indicates a lldp node is configured.";
      description "Disable LLDP TX or RX";
........................... (snipped) ............................
    description
      "This augment extends the configuration data of
      'Cisco-IOS-XR-ifmgr-cfg'";
  }
}
```

The structure of a data model can be explored using a YANG validator tool such as pyang and the data model can be formatted in a tree structure.

LLDP Configuration Data Model

The following example shows the LLDP interface manager configuration model in tree format.

```
module: Cisco-IOS-XR-ethernet-lldp-cfg
    +--rw lldp
       +--rw tlv-select!
       |  +--rw system-name
       |  |  +--rw disable?   boolean
       |  +--rw port-description
       |  |  +--rw disable?   boolean
       |  +--rw system-description
       |  |  +--rw disable?   boolean
       |  +--rw system-capabilities
       |  |  +--rw disable?   boolean
       |  +--rw management-address
       |  |  +--rw disable?   boolean
       |  +--rw tlv-select-enter       boolean
       +--rw holdtime?               uint32
       +--rw enable-priority-addr?   boolean
       +--rw extended-show-width?    boolean
       +--rw enable-subintf?         boolean
       +--rw enable-mgmtintf?        boolean
       +--rw timer?                  uint32
       +--rw reinit?                 uint32
       +--rw enable?                 boolean
module: Cisco-IOS-XR-ifmgr-cfg
    +--rw global-interface-configuration
    |  +--rw link-status?   Link-status-enum
    +--rw interface-configurations
       +--rw interface-configuration* [active interface-name]
          +--rw dampening
          |  +--rw args?                 enumeration
          |  +--rw half-life?            uint32
          |  +--rw reuse-threshold?      uint32
          |  +--rw suppress-threshold?   uint32
          |  +--rw suppress-time?        uint32
          |  +--rw restart-penalty?      uint32
          +--rw mtus
          |  +--rw mtu* [owner]
          |     +--rw owner    xr:Cisco-ios-xr-string
          |     +--rw mtu      uint32
          +--rw encapsulation
          |  +--rw encapsulation?         string
          |  +--rw capsulation-options?   uint32
          +--rw shutdown?                      empty
          +--rw interface-virtual?             empty
          +--rw secondary-admin-state?         Secondary-admin-state-enum
          +--rw interface-mode-non-physical?   Interface-mode-enum
          +--rw bandwidth?                     uint32
          +--rw link-status?                   empty
          +--rw description?                   string
```

```
         +--rw active                          Interface-active
         +--rw interface-name                  xr:Interface-name
         +--rw ethernet-lldp-cfg:lldp!
            +--rw ethernet-lldp-cfg:transmit
            |  +--rw ethernet-lldp-cfg:disable?   boolean
            +--rw ethernet-lldp-cfg:receive
            |  +--rw ethernet-lldp-cfg:disable?   boolean
            +--rw ethernet-lldp-cfg:lldp-intf-enter    boolean
            +--rw ethernet-lldp-cfg:enable?            Boolean

 ........................ (snipped) ........................
```

LLDP Operational Data Model

The following example shows the Link Layer Discovery Protocol (LLDP) interface manager operational model in tree format.

```
$ pyang -f tree Cisco-IOS-XR-ethernet-lldp-oper.yang
module: Cisco-IOS-XR-ethernet-lldp-oper
    +--ro lldp
       +--ro global-lldp
       |  +--ro lldp-info
       |     +--ro chassis-id?            string
       |     +--ro chassis-id-sub-type?   uint8
       |     +--ro system-name?           string
       |     +--ro timer?                 uint32
       |     +--ro hold-time?             uint32
       |     +--ro re-init?               uint32
       +--ro nodes
          +--ro node* [node-name]
             +--ro neighbors
             |  +--ro devices
             |  |  +--ro device*

........................ (snipped) ........................

  notifications:
    +---n lldp-event
       +--ro global-lldp
       |  +--ro lldp-info
       |     +--ro chassis-id?            string
       |     +--ro chassis-id-sub-type?   uint8
       |     +--ro system-name?           string
       |     +--ro timer?                 uint32
       |     +--ro hold-time?             uint32
       |     +--ro re-init?               uint32
       +--ro nodes
          +--ro node* [node-name]
             +--ro neighbors
             |  +--ro devices
             |  |  +--ro device*
             |  |     +--ro device-id?        string
             |  |     +--ro interface-name?   xr:Interface-name
             |  |     +--ro lldp-neighbor*
             |  |        +--ro detail
             |  |        |  +--ro network-addresses
             |  |        |  |  +--ro lldp-addr-entry*
             |  |        |  |     +--ro address
 ........................ (snipped) ........................
           +--ro interfaces
             |  +--ro interface* [interface-name]
             |     +--ro interface-name             xr:Interface-name
             |     +--ro local-network-addresses
             |     |  +--ro lldp-addr-entry*
             |     |     +--ro address
```

```
                |     |     |  +--ro address-type?   Lldp-l3-addr-protocol
                |     |     |  +--ro ipv4-address?   inet:ipv4-address
                |     |     |  +--ro ipv6-address?   In6-addr
                |     |     +--ro ma-subtype?   uint8
                |     |     +--ro if-num?       uint32
                |     +--ro interface-name-xr?         xr:Interface-name
                |     +--ro tx-enabled?                uint8
                |     +--ro rx-enabled?                uint8
                |     +--ro tx-state?                  string
                |     +--ro rx-state?                  string
                |     +--ro if-index?                  uint32
                |     +--ro port-id?                   string
                |     +--ro port-id-sub-type?          uint8
                |     +--ro port-description?          string
 ......................... (snipped) ........................
```

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:

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

- **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

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

Access the Data Models

You can access the Cisco IOS XR native and OpenConfig data models from GitHub, a software development platform that provides hosting services for version control.

CLI-based YANG data models, also known as unified configuration models were introduced in Cisco IOS XR, Release 7.0.1. The new set of unified YANG config models are built in alignment with the CLI commands.

You can also access the supported data models from the router. The router ships with the YANG files that define the data models. Use NETCONF protocol to view the data models available on the router using `ietf-netconf-monitoring` request.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
 <get>
   <filter type="subtree">
     <netconf-state xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
     <schemas/>
     </netconf-state>
```

```
   </filter>
 </get>
</rpc>
```

All the supported YANG models are displayed as response to the RPC request.

```
<rpc-reply message-id="16a79f87-1d47-4f7a-a16a-9405e6d865b9"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<data>
<netconf-state xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
<schemas>
<schema>
    <identifier>Cisco-IOS-XR-crypto-sam-oper</identifier>
    <version>1.0.0</version>
    <format>yang</format>
    <namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-crypto-sam-oper</namespace>
    <location>NETCONF</location>
</schema>
<schema>
    <identifier>Cisco-IOS-XR-crypto-sam-oper-sub1</identifier>
    <version>1.0.0</version>
    <format>yang</format>
    <namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-crypto-sam-oper</namespace>
    <location>NETCONF</location>
</schema>
<schema>
    <identifier>Cisco-IOS-XR-snmp-agent-oper</identifier>
    <version>1.0.0</version>
    <format>yang</format>
    <namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-snmp-agent-oper</namespace>
    <location>NETCONF</location>
</schema>

------------<snipped>--------------
<schema>
    <identifier>openconfig-aft-types</identifier>
    <version>1.0.0</version>
    <format>yang</format>
    <namespace>http://openconfig.net/yang/fib-types</namespace>
    <location>NETCONF</location>
</schema>
<schema>
    <identifier>openconfig-mpls-ldp</identifier>
    <version>1.0.0</version>
    <format>yang</format>
    <namespace>http://openconfig.net/yang/ldp</namespace>
    <location>NETCONF</location>
</schema>
</schemas>
</netconf-state>
------------<truncated>--------------
```

Prevent Partial Pseudo-Atomic Committed Configurations

Table 1: Feature History Table

Feature Name	Release Information	Description
Prevent Partial Pseudo-Atomic Committed Configurations	Release 7.10.1	You can now prevent the partially-committed configurations on the router and thus ensure the system database and OpenConfig datastore stay in sync. This feature changes how the internal rollback error is handled when a pseudo-atomic commit fails. In such cases, the system database always rolls back the configuration in its datastore thereby ensuring that there is no partially-committed configuration. If there is still inconsistency, the system displays error messages to notify you of various internal rollback failure scenarios based on which you must take rectification action to re-synchronize the data.

Existing Pseudo-Atomic Commit Behavior

The default behavior in pseudo-atomic commit is that all changes must succeed for the entire commit operation to succeed. If any errors are found, none of the configuration changes take effect.

Thus if an error occurs in one or more of the configurations in a commit, other configurations which were already successfully processed as part of the commit process are reverted. An internal rollback mechanism takes effect and reverts the already successful configurations to their original state.

Occasionally, the internal rollback may fail, that is, the verifier process rejects the rollback configuration. To stay in-sync with the verifier, the system database also does not rollback the configuration. This leads to commit of the failed-to-rollback configurations and results with system having partial committed configuration.

You can view the partial configuration with **show config commit changes [commit_id]** and take necessary action to keep the system database in-sync with verifiers.

Enhanced Pseudo-Atomic Commit Behavior

From IOS XR Software Release 7.10.1 onwards, for XR OpenConfig support, the running configurations in OpenConfig datastore can only be updated atomically. When the pseudo-atomic commit fails and the verifier rejects a rollback, OpenConfig datastore and system database would be out of sync in the existing pseudo-atomic commit behavior. The OpenConfig datastore would contain no changes from the commit, whereas the system database would contain configurations that failed to be rolled back.

The enhanced pseudo-atomic commit feature changes the way the internal rollback error is handled after a pseudo-atomic commit fails. This ensures the system database and OpenConfig datastore database stay in sync.

When the verifier process fails the configuration during an internal rollback, system database displays an `ios` error message to warn about the verifier error. You must take rectification action and re-synchronize the verifier and the system database. A failure to notice the error message or failure to restart the verifier process results in an inconsistent or deceptive operation of the system. After a while, the rollback error would become untraceable and could manifest into more problems.

Following are the scenarios with examples, where the internal rollback error appears when a pseudo-atomic commit fails:

- When the verifier process rejects the configuration during an internal rollback, system database displays an error message and continues to update system database and instruct the verifier to apply the configuration.

```
%MGBL_VERIFIER-4-COMMIT_ROLLBACK_REJECTED
```

Example shows the name of the process which rejects the internal rollback:

```
%MGBL-VERIFIER-4-COMMIT_ROLLBACK_REJECTED : verify_process incorrectly rejected rollback
 of a failed commit to a previously accepted state.  The rollback change has been made
 anyway. (/cfg/gl/test/item1, 0x40828400)
```

- When there is a timeout in the verify event (system database does not receive response from verifier within 300 seconds), then system database displays an `ios` message to warn you about the verifier timeout error and continue to update system database and instruct the verifier to apply the configuration.

```
%MGBL_VERIFIER-4-COMMIT_ROLLBACK_TIMEOUT
```

Example shows the name of the process which timeout for the internal rollback:

```
%MGBL-VERIFIER-3-COMMIT_ROLLBACK_TIMEOUT : verify_process (jid 68368, 0/0/CPU0) took
too long to verify the rollback of a failed commit
(cfg/if/act/GigabitEthernet0_0_0_2/a/test/item3). The rollback change has been made
anyway.
```

- When the verifier process fails to apply the internal rollback configuration or when the apply callback timeout, then the system database displays an `ios` message to warn you about the rollback failure and how to rectify the error by restarting the verifier process.

```
%MGBL_VERIFIER-3-COMMIT_ROLLBACK_FAILED
```

Example shows the name of the process which failed the internal rollback:

```
%MGBL-VERIFIER-3-COMMIT_ROLLBACK_FAILED : verify_process failed to apply the rollback
of a failed commit (/cfg/gl/test/item1, 0x40828400) and may no longer operate as
configured.  The process need to be restarted to rectify the error.
```

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)
- RPC framework (gRPC) by Google

Note gRPC is supported only in 64-bit platforms.

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 on network devices. It uses an Extensible Markup Language (XML)-based data encoding for the configuration data, as well as protocol messages. You use a simple NETCONF RPC-based (Remote Procedure Call) mechanism to facilitate communication between a client and a server. To get started with issuing NETCONF RPCs to configure network features using data models

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. You 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. To get started with issuing NETCONF RPCs to configure network features using data models

Note gRPC is supported only in 64-bit platforms.

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.

The following table shows a list of actions. For the full list of supported actions, query the device or see the YANG Data Models Navigator.

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
```