Data Models

Data Models - Programmatic and Standards-based Configuration

Cisco IOS XR software supports the automation of configuration of multiple routers across the network using Data models. Configuring routers using data models overcomes drawbacks posed by traditional router management techniques.

CLIs are widely used for configuring a router and for obtaining router statistics. Other actions on the router, such as, switch-over, reload, process restart are also CLI-based. Although, CLIs are heavily used, they have many restrictions.

Customer needs are fast evolving. Typically, a network center is a heterogenous mix of various devices at multiple layers of the network. Bulk and automatic configurations need to be accomplished. CLI scraping is not flexible and optimal. Re-writing scripts many times, even for small configuration changes is cumbersome. Bulk configuration changes through CLIs are error-prone and may cause system issues. The solution lies in using data models - a programmatic and standards-based way of writing configurations to any network device, replacing the process of manual configuration. Data models are written in a standard, industry-defined language. Although configurations using CLIs are easier (more human-friendly), automating the configuration using data models results in scalability.

Cisco IOS XR supports the YANG data modeling language. YANG can be used with Network Configuration Protocol (NETCONF) to provide the desired solution of automated and programmable network operations.

YANG model

YANG is a data modeling language used to describe configuration and operational data, remote procedure calls and notifications for network devices. The salient features of YANG are:

- Human-readable format, easy to learn and represent
- Supports definition of operations
- Reusable types and groupings
- Data modularity through modules and submodules

- Supports the definition of operations (RPCs)
- Well-defined versioning rules
- Extensibility through augmentation

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

NETCONF and gRPC (Google Remote Procedute Call) provide a mechanism to exchange configuration and operational data between a client application and a router and the YANG models define a valid structure for the data (that is being exchanged).

Protocol	Transport	Encoding/ Decoding
NETCONF	SSH	XML
gRPC	HTTP/2	XML, JSON

Each feature has a defined YANG model. Cisco-specific YANG models are referred to as synthesized models. Some of the standard bodies, such as IETF , IEEE and Open Config, are working on providing an industry-wide standard YANG models that are referred to as common models.

Components of Yang model

A module defines a single data model. However, a module can reference definitions in other modules and submodules by using the **import** statement to import external modules or the **include** statement to include one or more submodules. A module can provide augmentations to another module by using the **augment** statement to define the placement of the new nodes in the data model hierarchy and the **when** statement to define the conditions under which the new nodes are valid. **Prefix** is used when referencing definitions in the imported module.

YANG models are available for configuring a feature and to get operational state (similar to show commands)

This is the configuration YANG model for AAA (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)
```

This is the operational YANG model for AAA (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 may include any number of sub-modules, but each sub-module may belong to only one module. The names of all standard modules and sub-modules must be unique.

Structure of Yang models

YANG data models can be represented in a hierarchical, tree-based structure with nodes, which makes them more easily understandable. YANG defines four nodes types. Each node has a name, and depending on the node type, the node might either define a value or contain a set of child nodes. The nodes types (for data modeling) are:

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

Data types

YANG defines data types for leaf values. These data types help the user in understanding the relevant input for a leaf.

Name	Description
binary	Any binary data
bits	A set of bits or flags

Name	Description
boolean	"true" or "false"
decimal64	64-bit signed decimal number
empty	A leaf that does not have any value
enumeration	Enumerated strings
identityref	A reference to an abstract identity
instance-identifier	References a data tree node
int (integer-defined values)	8-bit, 16-bit, 32-bit, 64-bit signed integers
leafref	A reference to a leaf instance
uint	8-bit, 16-bit, 32-bit, 64-bit unsigned intergers
string	Human-readable string
union	Choice of member types

Data Model and CLI Comparison

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

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

The options available using the CLI are defined as leaf-nodes in data models. The defined data types, indicated corresponding to each leaf-node, help the user to understand the required inputs.

gRPC

gRPC is a language-neutral, open source, RPC (Remote Procedute Call) system developed by Google. By default, it uses protocol buffers as the binary serialization protocol. It can be used with other serialization protocols as well such as JSON, XML etc. 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. Although Protobufs was the only format supported in the initial release, gRPC is extensible to other content types. The Protobuf binary data object in gRPC is transported using HTTP/2 (RFC 7540). HTTP/2 is a replacement for HTTP that has been optimized for high performance. HTTP/2 provides many powerful capabilities including bidirectional streaming, flow control, header compression and multi-plexing. gRPC builds on those features, adding libraries for application-layer flow-control, load-balancing and call-cancellation.

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 in which the structure of the data is defined by YANG models.

Cisco gRPC IDL

The protocol buffers interface definition language (IDL) is used to define service methods, and define parameters and return types as protocol buffer message types.

gRPC requests can be encoded and sent across to the router using JSON. gRPC IDL also supports the exchange of CLI.

For gRPC transport, gRPC IDL is defined in .proto format. Clients can invoke the RPC calls defined in the IDL to program XR. The supported operations are - Get, Merge, Delete, Replace. The gRPC JSON arguments are defined in the IDL.

```
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) {};

}
```

gRPC Operations

- oper get-config—Retrieves a configuration
- oper merge-config— Appends to an existing configuration
- oper delete-config—Deletes a configuration
- oper replace-config—Modifies a part of an existing configuration
- oper get-oper—Gets operational data using JSON
- oper cli-config—Performs a configuration
- oper showcmdtextoutput

gNOI for BERT

Table 1: Feature History

Feature Name	Release Information	Description
gNOI for BERT	Cisco IOS XR Release 24.4.1	Extensible Manageability Services (EMS) gNOI supports Bit Error Rate Testing (BERT) operations on NCS 1014 for the following remote procedure calls (RPCs): • StartBERT • StopBERT • GetBERTResults gNOI for BERT is a vendor agnostic open configuration method of enabling and testing network links through the Pseudo Random Binary Sequence (PRBS) feature.

gRPC Network Operations Interface (gNOI) defines a set of gRPC-based microservices for executing operational commands on network devices. Extensible Manageability Services (EMS) gNOI is the Cisco IOS XR implementation of gNOI.

gNOI uses gRPC as the transport protocol and the configuration is same as that of gRPC.

From R24.4.10R24.4.1, EMS gNOI supports Bit Error Rate Testing (BERT) operations on NCS 1014 for the following remote procedure calls (RPCs):

- StartBERT
- StopBERT
- GetBERTResults

Start a New BERT Session

StartBERT

Starts a new BERT operation for a set of ports. Each BERT operation is uniquely identified by an ID, which is given by the caller. The caller can then use this ID (as well as the list of the ports) either to stop the BERT operation or get the BERT results, or can perform both BERT operations.

```
rpc StartBERT(StartBERTRequest) returns(StartBERTResponse) {}
```

Request and response messages

```
RPC to 10.127.60.184:57400
RPC start time: 13:59:45.488759
RPC start time: 13:59:45.488777
per_port_request for  startbert  is
 interface {
```

```
  elem {
    name: "terminal-device"
  }
  elem {
    name: "logical-channels"
  }
  elem {
    name: "channel"
    key {
      key: "index"
      value: "4014"
    }
  }
}
prbs_polynomial: PRBS_POLYNOMIAL_PRBS31
test_duration_in_secs: 360

Diag.StartBert Response
test_bert3
[interface {
  elem {
    name: "terminal-device"
  }
  elem {
name: "logical-channels"
  }
  elem {
    name: "channel"
    key {
      key: "index"
      value: "4014"
    }
  }
}
status: BERT_STATUS_OK
]
RPC end time: 13:59:45.816653
RPC end time: 2024-11-05 08:29:45.816739
```

The supported values for **prbs_polynomial** on NCS1014:

- **Trunk Ports** — PRBS7, PRBS13, PRBS23, and PRBS31
- **Client Ports** — PRBS23 and PRBS31

The **StartBERT** RPC can return an error status in any one of the following scenarios:

- When BERT operation is supported on none of the ports specified by the request.
- When BERT is already in progress on any port specified by the request.
- In case of any low-level hardware or software internal errors.

The RPC returns an **OK** status when there is no error situation encountered.

Stop and Delete an Existing BERT Session from the Device

Stops an already in-progress BERT operation on a set of ports. The caller uses the BERT operation ID it previously used when starting the operation to stop it.

StopBERT

```
rpc StopBERT(StopBERTRequest) returns(StopBERTResponse) {}
```

Request and response messages

```
message StopBERTRequest {
RPC to 10.127.60.184:57400
RPC start time: 13:59:27.642444
RPC start time: 13:59:27.642462
per_port_request for  stopbert  is
 interface {
  elem {
    name: "terminal-device"
  }
  elem {
    name: "logical-channels"
  }
  elem {
    name: "channel"
    key {
      key: "index"
      value: "4014"
    }
  }
}

bert_operation_id: "test_bert3"
per_port_requests {
  interface {
    elem {
      name: "terminal-device"
    }
    elem {
      name: "logical-channels"
    }
    elem {
      name: "channel"
      key {
        key: "index"
        value: "4014"
      }
    }
  }
}

message StopBERTResponse {
  bert_operation_id: "test_bert3"
per_port_responses {
  interface {
    origin: "openconfig-terminal-device"
    elem {
      name: "terminal-device"
    }
    elem {
      name: "logical-channels"
    }
    elem {
      name: "channel"
      key {
        key: "index"
        value: "4014"
      }
```

```
    }
  }
  status: BERT_STATUS_OK
}

Diag.StopBert Response

test_bert3
[interface {
  origin: "openconfig-terminal-device"
  elem {
    name: "terminal-device"
  }
  elem {
    name: "logical-channels"
  }
  elem {
    name: "channel"
    key {
      key: "index"
      value: "4014"
    }
  }
}
status: BERT_STATUS_OK
]
RPC end time: 13:59:27.726083
RPC end time: 2024-11-05 08:29:27.726099

}
```

When the **PerPortRequest** field is not configured, then the device stops and deletes BERT sessions on all the ports associated with the BERT ID.

The RPC is expected to return an error status in any one of the following situations:

- When there is at least one BERT operation in progress on a port which cannot be stopped in the middle of the operation (either due to lack of support or internal problems).
- When no BERT operation, which matches the given BERT operation ID, is in progress or completed on any of the ports specified by the request.

The **StopBERT** RPC returns to an **OK** status when there is no error situation is encountered.

Note The BERT operation is considered completed if the device has a record or history of it. Also note that you might receive a stop request for a port which has completed BERT, as long as the recorded BERT operation ID matches the one specified by the request.

Get BERT Statistics for an Existing Session

Gets BERT results during the BERT operation or after the operation completes. The caller uses the BERT operation ID that it previously used when starting the operation to query it. The device stores results for the last BERT based on the required period of time.

GetBERTResults

```
rpc GetBERTResult(GetBERTResultRequest) returns(GetBERTResultResponse) {}
```

Request and response messages

```
message GetBERTResultRequest {
  RPC to 10.127.60.184:57400
RPC start time: 14:00:01.623902
RPC start time: 14:00:01.623919
per_port_request for  getbertresult  is
 interface {
  elem {
    name: "terminal-device"
  }
  elem {
    name: "logical-channels"
  }
  elem {
    name: "channel"
    key {
      key: "index"
      value: "4014"
    }
  }
}

message GetBERTResultResponse {
  test_bert3
[interface {
  elem {
    name: "terminal-device"
  }
  elem {
    name: "logical-channels"
  }
  elem {
    name: "channel"
    key {
      key: "index"
      value: "4014"
    }
  }
}
status: BERT_STATUS_OK
]
RPC end time: 13:59:45.816653
RPC end time: 2024-11-05 08:29:45.816739

}
```

When the **per_port_requests** is ignored, then the device returns results and status for all the ports associated with the BERT ID.

The following table lists the descriptions of BERT results and status.

Table 2: BERT Results and Status

Field	Description
interface	Port in **types.Path** format representing a path in the open configuration interface model.

Field	Description
status	• **BERT_STATUS_OK** denotes that the BERT session is active. • **BERT_STATUS_PORT_NOT_RUNNING_BERT** denotes that BERT is not running as the duration has expired. • **BERT_STATUS_NON_EXISTENT_PORT** denotes that specified port is not found. • **BERT_STATUS_UNSUPPORTED_PRBS_POLYNOMIAL** denotes that PRBS generating polynomial is not supported by the target. • **BERT_STATUS_PORT_ALREADY_IN_BERT** denotes that there is already a BERT running on the specified port. Returns when the StartBERT RPC attempts to initiate BERT on a port that is already in use. • **BERT_STATUS_OPERATION_ID_IN_USE** denotes that the specified BERT operation ID is already in use. This occurs when the StartBERT RPC attempts to use an ID that has already been assigned to an existing BERT operation. • **BERT_STATUS_OPERATION_ID_NOT_FOUND** denotes that the specified BERT operation ID is not recognized. This response is applicable for both StopBERT and GetBERTResult RPCs.
bert_operation_id	BERT operation ID that the port is associated with.
test_duration_in_secs	BERT duration in seconds. Must be a positive number.
prbs_polynomial	The PRBS polynomial value that is configured.
last_bert_start_timestamp	Start operation timestamp in form of a 64-bit value UNIX time, which is the number of seconds elapsed since January 1, 1970 UTC.
repeated last_bert_get_results_timestamp	Timestamp of the last GetBERTResults operation in form of a 64-bit value UNIX time, which is the number of seconds elapsed since January 1, 1970 UTC.
peer_lock_established	Current status of peer lock. Note that there could be a 10-second delay in updating this field.
peer_lock_lost	Indicates if the peer lock is lost anytime after a peer lock is established. This field is only meaningful if **peer_lock_established** field is set.

Field	Description
error_count_per_minute	A list of one-minute historical PM buckets containing bit error counts. Historical buckets are maintained since the StartBERT operation started.
total_errors	Cumulative count of bit errors of the StartBERT operation.

The GetBERTResults RPC can return error status in any one of the following scenarios:

- When no BERT operation, which matches the given BERT operation ID, is in-progress or completed on any of the ports specified by the request.
- When the BERT operation ID does not match the in progress or completed BERT operation on any of the ports specified by the request.

The RPC returns an **OK** status when none of these situations is encountered.

Note The BERT operation is considered as completed only when the device has a record of it.

gNOI Healthz service

Table 3: Feature History

Feature Name	Release Information	Feature Description
gNOI Healthz	Cisco IOS XR Release 25.2.1	The gNOI Healthz sub-functions monitor and manage the state of a device by leveraging telemetry within the network. These sub-functions help determine the device's system state and facilitate the collection of relevant debug logs based on that state, using the OpenConfig Healthz model.

The gNOI Healthz is a gRPC service that focuses on the health checks and monitoring of network devices. It determines whether the nodes of a network are fully functional, degraded, or need to be replaced. The gNOI Healthz process involves:

- Waiting for health status data from various subsystem components
- Inspecting and analyzing health status data to identify any unhealthy entities, and
- Collecting the debug logs of the corresponding unhealthy component

gNOI Healthz, in conjunction with gNMI telemetry, monitors the health of network components.

When a component becomes `HEALTHY` or `UNHEALTHY`, telemetry updates are sent for that health event. For more details about the health event, see gNOI Healthz RPCs. When a system component changes its state to

UNHEALTHY, the intended artifacts (debug logs, core file, and so on) are generated automatically at the time of the health event.

XR health event dampening is enabled by default to manage and stabilize health-event behavior. It works by suppressing excessive state changes or flapping events to ensure the system remains stable.

Supported components

Hardware:

- Rack 0
- 0/RP0/CPU0
- Line cards

See Healthz hardware events list, on page 14.

Device health status updates workflow

This section describes the workflow for monitoring and managing device component health status using gNOI Healthz RPCs and telemetry.

1. The client subscribes to the component's OpenConfig path with an ON_CHANGE request and waits for a health event to occur. When a health event is detected in the device for that component, the client receives a notification. The client monitors these health parameters:
   - **status**: HEALTHY, UNHEALTHY, or UNKNOWN
   - **last-unhealthy**: Timetsamp of last known healthy state
   - **unhealthy-count**: Number of times the particular component is reported unhealthy
2. When the device receives gNOI Healthz RPCs from gNOI client, it performs these actions and responds to the gNOI client.

Table 4: gNOI Healthz RPCs

When the gNOI client sends...	The device...
Get RPC	retrieves the latest set of health statuses that are associated with a specific component and its subcomponents.
List RPC	returns all events that are associated with a device.
Artifact RPC	retrieves specific artifacts that are listed by the target system in the List() or Get() RPC.
Acknowledge RPC	acknowledges a series of artifacts that are listed by the Acknowledge() RPC.
Check RPC	performs intensive health checks that may impact the service, ensuring they are done intentionally to avoid disruptions.

Examples of heath status transmitted through EDT

When a system component becomes unhealthy, the system transmits health state information via telemetry, indicating that the healthz/state/status of the component has transitioned to **UNHEALTHY**. An example EDT notification received by the client is:

```
{
  "source": "10.127.60.184:57400",
  "subscription-name": "default-1748690348",
  "timestamp": 1748690279018280149,
  "time": "2025-05-31T16:47:59.018280149+05:30",
  "prefix": "openconfig:",
  "updates": [
    {
      "Path": "components/component[name=Rack 0]/healthz",
      "values": {
        "components/component/healthz": {
          "state": {
            "last-unhealthy": "1748690279018280149",
            "status": "UNHEALTHY",
            "unhealthy-count": "8"
          }
        }
      }
    }
  ]
}
```

An example of EDT notification received by the client, when a system component becomes healthy:

```
{
  "source": "192.0.1.0:57400",
  "subscription-name": "default-1748690348",
  "timestamp": 1748690408752434448,
  "time": "2025-05-31T16:50:08.752434448+05:30",
  "prefix": "openconfig:",
  "updates": [
    {
      "Path": "components/component[name=Rack 0]/healthz",
      "values": {
        "components/component/healthz": {
          "state": {
            "last-unhealthy": "1748690279018280149",
            "status": "HEALTHY",
            "unhealthy-count": "8"
          }
        }
      }
    }
  ]
}
```

Healthz hardware events list

This table lists the hardware events that can lead to an unhealthy health status.

Table 5: Healthz hardware events list

Alarm Name	OC_COMP	Health Status
FAN-TRAY-ABSENT	Rack 0	Unhealthy

Alarm Name	OC_COMP	Health Status
CPU_NOT_SEATED_PROPERLY_FAILURE	Rack 0	Unhealthy
CPU_PRESENCE_PIN_FAILURE	Rack 0	Unhealthy
CPU-FPGA-PCIE-ERROR	0/RP0/CPU0	Unhealthy
PHY1-MDIO-ACCESS-ERROR	0/RP0/CPU0	Unhealthy
EITU-FPGA-PCIE-ERROR	0/RP0/CPU0	Unhealthy
FAM_FAULT_TAG_LC_METADX1_FAILURE0 **FAM_FAULT_TAG_LC_METADX1_FAILURE1**	Line card Location	Unhealthy
FAM_FAULT_TAG_LC_RAM_FAILURE	Line card Location	Unhealthy
FAM_FAULT_TAG_LC_FPGA_FAILURE	Line card Location	Unhealthy
FAM_FAULT_TAG_LC_BOARD_IO_FAILURE	Line card Location	Unhealthy
FAM_FAULT_TAG_LC_VIRTUAL_WIRE_FAILURE0 **FAM_FAULT_TAG_LC_VIRTUAL_WIRE_FAILURE1**	Line card Location	Unhealthy
FAM_FAULT_TAG_LC_PLL_FAILURE	Line card Location	Unhealthy
FAM_FAULT_TAG_LC_METADX2_FAILURE0 **FAM_FAULT_TAG_LC_METADX2_FAILURE1** **FAM_FAULT_TAG_LC_METADX2_FAILURE2** **FAM_FAULT_TAG_LC_METADX2_FAILURE3** **FAM_FAULT_TAG_LC_METADX2_FAILURE4**	Line card Location	Unhealthy
FAM_FAULT_TAG_LC_CPU_CORRUPTION	Line card Location	Unhealthy
FAM_FAULT_TAG_LC_FPD_FAILURE	Line card Location	Unhealthy
FAM_FAULT_TAG_CRYPTO_HW_FAILURE	Line card Location	Unhealthy
FAM_FAULT_TAG_LC_BOARD_IO1_FAILURE	Line card Location	Unhealthy

Verify device health using gNOI RPCs

Follow these steps to monitor health status telemetry of a device using gNOI healthz RPC.

Procedure

Step 1 Monitor health state of the device.

Example:

```
RP/0/RP0/CPU0:ios#show health status
Wed Jul 24 10:03:29.811 UTC
SNo   Component name                          Health status
----- ------------------------------------- --------------------
1     0_RP0_CPU0-appmgr                     healthy
2     0_RP0_CPU0-ownershipd                 healthy
RP/0/RP0/CPU0:ios#
RP/0/RP0/CPU0:ios#show health status 0_RP0_CPU0-appmgr
Wed Jul 24 10:03:46.859 UTC
Sno   Event Id            Timestamp           Status     Artifacts
----- ------------------- ------------------- ---------- --------------------------------
1     1721815321290718105 Jul 24 10:02:01 UTC healthy    []

2     1721815320614225976 Jul 24 10:02:00 UTC unhealthy  ['/harddisk:/eem_ac_logs/xrhealth
                                                         /artifacts/procmgr_event_20240724
                                                         100205.tar.gz', '/misc/disk1/appm
                                                         gr_8921.by.11.20240724-100159.nod
                                                         e0_RP0_CPU0.09b9c.core.gz']
```

Step 2 Monitor device health with gNOI **Get RPC**.

Example:

```
ios#/auto/appmgr/xrhealth/bin/gnoic -a 192.0.2.184:57400 --insecure -u cisco -p cisco123  healthz
get --path "openconfig:/components/component[name=${OC_COMP}]"
WARN[0000] "192.0.2.184:57400" path     : openconfig:components/component[name=Rack 0]
  status   : STATUS_UNHEALTHY
  id       : 1748850022419622614
  acked    : false
  created  : 2025-06-02 07:40:22.419622614 +0000 UTC
  expires  : 2025-06-09 07:40:22.041962261 +0000 UTC
  artifict :
    - id       : Rack
0-1748850022419622614-704c7b084eb499ade4fb9c8636fd70fc34cb683b344625ee20e46f7b79b312ad
      name     : Rack_0_SYSTEM_HW_ERROR_FAM_FAULT_TAG_FAN_TRAY_ABSENT_20250602131022.tar
      path     :
/harddisk:/eem_ac_logs/xrhealth/artifacts/Rack_0_SYSTEM_HW_ERROR_FAM_FAULT_TAG_FAN_TRAY_ABSENT_20250602131022.tar
      mimeType : application/x-tar
      size     : 30720
      hash     : SHA256(7cc93c4c829f5899552d8bb8449b5e4662f8ffdbd8d43636ef2446871349e613)

================================================================================

  path     : openconfig:components/component[name=Rack 0]
  status   : STATUS_HEALTHY
  id       : 1748698891342716588
  acked    : false
  created  : 2025-05-31 13:41:31.342716588 +0000 UTC
  expires  : 2025-06-07 13:41:31.003427167 +0000 UTC
```

Step 3 Monitor device health with gNOI **Check RPC**.

Example:

```
ios#/auto/appmgr/xrhealth/bin/gnoic -a 192.0.2.184:57400 --insecure -u cisco -p cisco123  healthz
check --path "openconfig:/components/component[name=${OC_COMP}]"
WARN[0000] "192.0.2.184:57400" could not lookup hostname: lookup 198.51.100.10.in-addr.arpa. on
64.104.128.236:53: no such host
target "192.0.2.184:57400":
+-----------------+--------------------+-------------------------------------------+--------+
|   Target Name   |         ID         |                   Path                    |
Status    |              Created At              |
Artifact ID                                 | Artifact Type |
+-----------------+--------------------+-------------------------------------------+--------+
| 192.0.2.184:57400 | 1748850219768107864 | openconfig:components/component[name=Rack 0] |
```

```
STATUS_UNHEALTHY | 2025-06-02 07:43:39.769145114 +0000 UTC |
 Rack 0-1748850219768107864-348a1a481f4b464d72c2f90648c85d2870525943f601aee745649c1e3fb8102c | file
         |
+--------+--------+-------------------+--------+-----------------+-------------------------------------------------------------------+------+
```

On the NCS1014, the Check RPC collects the "show tech ncs10xx" file as an artifact. The artifact is stored in the system directory: /harddisk:/eem_ac_logs/xrhealth/artifacts/.

The Check RPC request typically takes 15-20 minutes to complete.

A CheckRequest for a previous event_id does not overwrite artifacts collected during the event time.

Step 4 Monitor device health with gNOI **List RPC**.

Example:

```
ios#/auto/appmgr/xrhealth/bin/gnoic -a 192.0.2.184:57400 --insecure -u cisco -p cisco123  healthz
list --path "openconfig:/components/component[name=${OC_COMP}]"
WARN[0000] "192.0.2.184:57400" could not lookup hostname: lookup 198.51.100.10.in-addr.arpa. on
64.104.128.236:53: no such host
target "192.0.2.184:57400":
+--------+--------+-------------------+--------+-----------------+-------------------------------------------------------------------+
|     Target Name     |         ID          |                    Path                     |
Status      |              Created At               |
Artifact ID                                      |
+--------+--------+-------------------+--------+-----------------+-------------------------------------------------------------------+
| 192.0.2.184:57400 | 1748690279018280149 | openconfig:components/component[name=Rack 0] |
STATUS_UNHEALTHY | 2025-05-31 11:17:59.018280149 +0000 UTC | Rack 0-
1748690279018280149-12a0fa57b1624baae49f742a791d3017ecf219eeb32fb9187a3f9602d0856ba5 |
| 192.0.2.184:57400 | 1748690408752434448 | openconfig:components/component[name=Rack 0] |
STATUS_HEALTHY   | 2025-05-31 11:20:08.752434448 +0000 UTC   |

| 192.0.2.184:57400 | 1748850022419622614 | openconfig:components/component[name=Rack 0] |
STATUS_UNHEALTHY | 2025-06-02 07:40:22.419622614 +0000 UTC | Rack 0-
1748850022419622614-704c7b084eb499ade4fb9c8636fd70fc34cb683b344625ee20e46f7b79b312ad |
| 192.0.2.184:57400 | 1748850219768107864 | openconfig:components/component[name=Rack 0] |
STATUS_UNHEALTHY | 2025-06-02 07:43:39.769145114 +0000 UTC | Rack 0-
1748850219768107864-348a1a481f4b464d72c2f90648c85d2870525943f601aee745649c1e3fb8102c |
+--------+--------+-------------------+--------+-----------------+-------------------------------------------------------------------+


ios#/auto/appmgr/xrhealth/bin/gnoic -a 192.0.2.184:57400 --insecure -u cisco -p cisco123  healthz
list --path "openconfig:/components/component[name=${OC_COMP}]" --acked
WARN[0000] "192.0.2.184:57400" could not lookup hostname: lookup 198.51.100.10.in-addr.arpa. on
64.104.128.236:53: no such host
target "192.0.2.184:57400":
+--------+--------+-------------------+--------+-----------------+-------------------------------------------------------------------+
|     Target Name     |         ID          |                    Path                     |
Status      |              Created At               |
Artifact ID                                      |
+--------+--------+-------------------+--------+-----------------+-------------------------------------------------------------------+
| 192.0.2.184:57400 | 1748350110127223712 | openconfig:components/component[name=Rack 0] |
STATUS_UNHEALTHY | 2025-05-27 12:48:30.127223712 +0000 UTC |
Rack 0-1748350110127223712-3c2027c2f3a0f3b86d7e75ffb394807aab9c87f5d47e153f9c40dc78f66b7227 |
| 192.0.2.184:57400 | 1748350110127223712 | openconfig:components/component[name=Rack 0] |
STATUS_UNHEALTHY | 2025-05-27 12:48:30.127223712 +0000 UTC |
Rack 0-1748350110127223712-067f9b0ef240f92a0b19905306d8dd53dc699574d3720452bde39e630942b8e7 |
| 192.0.2.184:57400 | 1748351569014046087 | openconfig:components/component[name=Rack 0] |
STATUS_HEALTHY   | 2025-05-27 13:12:49.014046087 +0000 UTC |
                                                 |
| 192.0.2.184:57400 | 1748417625250765686 | openconfig:components/component[name=Rack 0] |
STATUS_UNHEALTHY | 2025-05-28 07:33:45.250765686 +0000 UTC |
Rack 0-1748417625250765686-799aebceab6827ec62fa65521ab927738fea2d75f0210d230026c3694831e781 |
| 192.0.2.184:57400 | 1748418605270788333 | openconfig:components/component[name=Rack 0] |
```

```
STATUS_HEALTHY   | 2025-05-28 07:50:05.270788333 +0000 UTC |
                                                           |
```

Step 5 Monitor device health with gNOI **Artifact RPC**.

Example:

```
ios#/auto/appmgr/xrhealth/bin/gnoic -a 192.0.2.184:57400 --insecure -u cisco -p cisco123  healthz
artifact --id "Rack
0-17488502197681078 64-348a1a481f4b464d72c2f90648c85d2870525943f601aee745649c1e3fb8102c"
WARN[0000] "192.0.2.184:57400" could not lookup hostname: lookup 198.51.100.10.in-addr.arpa. on
64.104.128.236:53: no such host
INFO[0003] 192.0.2.184:57400: received file header for artifactID: Rack
0-17488502197681078 64-348a1a481f4b464d72c2f90648c85d2870525943f601aee745649c1e3fb8102c
id:  "Rack 0-17488502197681078 64-348a1a481f4b464d72c2f90648c85d2870525943f601aee745649c1e3fb8102c"
file:  {
  name:  "showtech-dt_healthz-ncs10xx-2025-Jun-02.131340.IST.tgz"
  path:
"/harddisk:/eem_ac_logs/xrhealth/artifacts/showtech-dt_healthz-ncs10xx-2025-Jun-02.131340.IST.tgz"
  mimetype:  "application/octet-stream"
  size:  138887423
  hash:  {
    method:  SHA256
    hash:  "vU5\x85l@\xd32\x1b\xe4\xd5w\xfbc\x07\x18w\x96= \xbc\xd1\x13Y\x94\xba\x1e-w\x1b\xda\xfc"
  }
}

INFO[0003] received 65536 bytes for artifactID: Rack
0-17488502197681078 64-348a1a481f4b464d72c2f90648c85d2870525943f601aee745649c1e3fb8102c
INFO[0007] received 16639 bytes for artifactID: Rack
0-17488502197681078 64-348a1a481f4b464d72c2f90648c85d2870525943f601aee745649c1e3fb8102c
INFO[0007] 192.0.2.184:57400: received trailer for artifactID: Rack
0-17488502197681078 64-348a1a481f4b464d72c2f90648c85d2870525943f601aee745649c1e3fb8102c
INFO[0007] 192.0.2.184:57400: received 138887423 bytes in total
INFO[0007] 192.0.2.184:57400: comparing file HASH
INFO[0007] 192.0.2.184:57400: HASH OK
ios$
```

Step 6 Monitor device health with gNOI **Acknowledge RPC**.

Example:

```
ios#/auto/appmgr/xrhealth/bin/gnoic -a 192.0.2.184:57400 --insecure -u cisco -p cisco123  healthz
ack --path "openconfig:/components/component[name=${OC_COMP}]" --id 1748853831199330797
WARN[0000] "192.0.2.184:57400" could not lookup hostname: lookup 198.51.100.10.in-addr.arpa. on
64.104.128.236:53: no such host

target "192.0.2.184:57400":
  path     : openconfig:components/component[name=Rack 0]
  status   : STATUS_UNHEALTHY
  id       : 1748853831199330797
  acked    : true
  created  : 2025-06-02 08:43:51.199330797 +0000 UTC
  expires  : 2025-06-09 08:43:51.019933079 +0000 UTC
  artifict :
    - id       : Rack
0-1748853831199330797-74773059551bb85a0629a6ac8e00aacf3c65af1530033c9d74909194243a2c08
      name     : Rack_0_SYSTEM_HW_ERROR_FAM_FAULT_TAG_FAN_TRAY_ABSENT_20250602141351.tar
      path     :
/harddisk:/eem_ac_logs/xrhealth/artifacts/Rack_0_SYSTEM_HW_ERROR_FAM_FAULT_TAG_FAN_TRAY_ABSENT_20250602141351.tar
      mimeType : application/x-tar
      size     : 30720
      hash     : SHA256(051c2d0325e11ec0efb18521e76a3d53d2f6153813f90f0d240d56769a6f511f)
```

```
ios$
```