NETMOD Q. Ma, Ed.
Internet-Draft C. Feng
Updates: RFC6241, RFC8040, RFC8342 (if approved) Q. Wu
Intended status: Standards Track Huawei
Expires: 28 April 2022 25 October 2021

System-defined Configuration
draft-ma-netmod-with-system-00

Abstract

This document updates NMDA [RFC 8342] to define a read-only conventional configuration datastore called "system" to hold system-defined configurations. To support non-NMDA servers, a "with-system" parameter has been defined to return <running> and system-defined configuration combined. The solution enables clients to reference nodes defined in <system>, overwrite values of configurations defined in <system>, and configure descendant nodes of system-defined nodes.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

NMDA Architecture [RFC8342] defines system configuration as the configuration that is supplied by the device itself and should be present in <operational> when it is in use.

However, there is a desire to enable a server to better document the system configuration. Clients can benefit from a standard mechanism to see what system configuration is available in a server.

In some cases, a client or offline tool may consider the configuration in <running> or <intended> invalid due to references (e.g. leafref) to system configuration data that isn't returned when the datastore is read. The server may accept a configuration (i.e. by internally merging the client specified contents of <running> with the server-provided system configuration and validating the result), but the client or offline tool would consider the datastore contents as invalid.

Having to copy the entire contents of the system configuration into <running> should be avoided or reduced when possible.

In some other cases, configuration of descendant nodes of system defined configuration needs to be supported. For example, the system configuration may contain an almost empty physical interface, while the client needs to be able to add, modify, remove a number of descendant nodes. Some descendant nodes may not be modifiable (e.g. "name" and "type" set by the system).

In all cases, the clients should have control over the configurations ,i.e., read-back of <running> should contain only what was explicitly set by clients.

This document updates NMDA [RFC 8342] to define a read-only conventional configuration datastore called "system" to hold system-defined configurations. To support non-NMDA servers, a "with-system" parameter has been defined to return <running> and system-defined configuration combined. The solution enables clients to reference nodes defined in <system>, overwrite values of configurations defined in <system>, and configure descendant nodes of system-defined nodes.

1.1. Terminology

This document assumes that the reader is familiar with the contents of [RFC6241], [RFC7950], [RFC8342], [RFC8407], and [RFC8525] and uses terminologies from those documents.

The following terms are defined in this document as follows:

System configuration: Configuration that is provided by the system itself [RFC8342].

Conventional configuration datastore: One of the following set of configuration datastores: <running>, <startup>, <candidate>, <system>, and <intended>. These datastores share a common datastore schema, and protocol operations allow copying data between these datastores. The term "conventional" is chosen as a generic umbrella term for these datastores.

System configuration datastore: A configuration datastore holding the complete configuration provided by the system itself. This datastore is referred to as "<system>".

1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.3. Updates to RFC 6241

The <get> and <get-config> RPC operations defined in [RFC6241] are augmented to accept additional new input parameter "with-system" which carries no value. The retrieval of implicit hidden system configuration in <running> can be used through <get> or <get-config> operation with the presence of "with-system" parameter.

The implicit hidden system configuration will contain all three types of system configurations defined in Section 2.

Note that the <get-data> RPC operation defined in [RFC8526] can also be augmented to retrieve the system configuration from <running>. But not sure whether the new client only supports <get-data> operation or supports both <get-config> operation and <get-data> operation.

1.4. Updates to RFC 8040

This document extends Section 4.8 of [RFC8040] to add a new query parameter "with-system".

The "with-system" parameter controls whether implicitly hidden system configuration will be returned in the reply. This parameter is only allowed with no values carried. If this parameter has any unexpected value, then a "400 Bad Request" status-line is returned.

Name	Methods	Description
with-system	GET, HEAD	indicates that the implicitly hidden system configuration should be returned. If not specified, then no implicitly hidden system configuration should be returned. This parameter can be given in any order.

2. Kinds of System Configuration

There are three types of system configurations: immediately-activated system configuration, conditionally-activated system configuration and inactivated-until-referenced system configuration.

2.1. Immediately-Active

Immediately-active system configurations are those applied and active immediately (e.g., a loop-back interface) , irrespective of physical resource present or not, a special functionality enabled or not.

2.2. Conditionally-Active

System configurations which are provided and activated based on specific conditions being met in a system, e.g.,if a physical resource is present (e.g., insert interface card), the system will automatically detect it and load pre-provisioned configuration; when the physical resource is not present(remove interface card), the system configuration will be automatically cleared. Another example is when a special functionality is enabled, e.g., when QoS function is enabled, QoS policies are automatically created by the system.

2.3. Inactive-Until-Referenced

There are some predefined objects(e.g., application ids, anti-x signatures, trust anchor certs, etc) as a convenience for the clients. The clients can also define their own data objects for their unique requirements. Inactive-until-referenced system configurations are not applied and active immediately but only after they are referenced by client defined configuration.

3. Static Characteristics

3.1. Read-only to Clients

From the clients' perspective, the contents of the <system> datastore are read-only. There is no way to delete system configuration from a server. Any deletable system-provided configuration must be defined in <factory-default> [RFC 8808], which is used to initialize <running> when the device is first-time powered on or reset to its factory default condition.

3.2. May Change via Software Upgrades

System configuration MAY change dynamically, e.g., depending on factors like during device upgrade or system-controlled resources(e.g., HW available) . In some implementations, when QoS function is enabled, QoS-related predefined policies are created by system. If the system configuration gets changed, YANG notification (e.g., "push-change-update" notification)[RFC8641][RFC8639][RFC6470] can be used to notify the client.

3.3. No Impact to <operational>

This work intends to have no impact to <operational>. As always, system configuration will appear in <operational> with "origin=system". This work enables a subset of those system generated nodes to be defined like configuration, i.e., made visible to clients in order for being referenced or configurable prior to present in <operational>. "Config false" nodes are completely out of scope, hence existing "config false" nodes are not impacted by this work.

4. Dynamic Behavior

4.1. Conceptual Model

This document introduces an optional datastore named "system" which is used to hold all three types of system configurations defined in Section 2.

When the device is powered on, immediately-activated system configuration will be provided and activated immediately but inactivated-until-referenced system configuration only becomes active if it is referenced by client defined configuration. While conditionally-activated system configuration will be created and immediately activated if the condition on system resources is met when the device is powered on or running.

All these system configuration will be implicitly hidden in the <running>, hence the client can retrieve them through standard operations defined in YANG-driven management protocols such as NETCONF and RESTCONF with a "with-system" query parameter. So that the client can get a merged view from the server.

If the <system> datastore exists, all above three types of system configurations will also go into <system>. Then the server will merge <running> and <system> to create <intended>, in which process, <running> MAY overwrite and/or extend <system>. If a server implements <intended>, <system> MUST be merged into <intended>.

When the client needs to configure the descendant nodes of system configuration(e.g., a physical interfaces), the ancestor system configuration needs to be configured in <running> explicitly.

4.2. Modifying (overriding) system configuration

In some cases, a server may allow some parts of system configuration to be modified. List keys in system configuration can't be changed by a client, but other descendant nodes in a list entry may be modifiable or non-modifiable. Leafs and leaf-lists outside of lists may also be modifiable or non-modifiable. Modification of system configuration is achieved by the client writing configuration to <running> that overrides the system configuration. Client configuration statements in <running> take precedence over system configuration nodes in <system> if the server allows the nodes to be modified. If a system configuration node is non-modifiable, then writing a value for that node in <running> returns an error.

A server may also allow a client to add data nodes to a list entry in <system> by writing those additional nodes in <running>. Those additional data nodes may not exist in <system> (i.e. an *addition* rather than an override).

While modifying (overriding) system configuration nodes may be supported by a server, there is no mechanism for deleting a system configuration node. A "mandatory true" leaf, for example, may have a value in <system> which can be modified (overridden) by a client setting that leaf to a value in <running>. But the leaf could not be deleted.

Comment 1: What if <System> contains a set of values for a leaf-list, and a client configures another set of values for that leaf-list in <running>, will the set of values in <running> completely replace the set of values in <system>? Or the two sets of values are merged together?

Comment 2: how "ordered-by user" lists and leaf-lists are merged? Do the <running> values go before or after, or is this a case where a full-replace is needed.

4.3. Explicit declaration of system configuration

In addition to modifying system configuration, and adding nodes to lists in system configuration as described above, a client can also explicitly declare system configuration nodes in <running> with the same values as in <system>. When a client configures a node (list entry, leaf, etc) in <running> that matches the same node & value in <system>, then that node becomes part of <running>. A read of <running> returns those explicitly configured nodes.

This explicit configuration of system configuration in <running> can be useful, for example, when an operator's workflow requires a client or offline tool to see the <running> configuration as valid. The client can explicitly declare (i.e. configure in <running>) the list entries (with at least the keys) for any system configuration list entries that are referenced elsewhere in <running>. The client does not necessarily need to declare all the contents of the list entry (i.e. the descendant nodes) - only the parts that are required to make the <running> appear valid offline.

4.4. Examples

The examples within this document use the fictional interface YANG module defined in Appendix C.3 of [RFC8342]. In addition, a fictional QoS data model example is provided.

4.4.1. Modifying A System-instantiated Leaf's Value

In this subsection, we will use this fictional QoS data model:

```
container qos-policies {
   list policy {
     key "name";
     leaf name {
     type string;
   }
     list queue {
       key "queue-id";
       description "Enter the queue list instance";
         leaf queue-id {
           type int32 {
             range "1..32";
           }
         }
         leaf maximum-burst-size {
           type int32 {
             range "0..100";
           }
         }
       }
     }
   }
```

Suppose a client creates a qos policy "my-policy" with 4 system instantiated queues(1~4). The Configuration of qos-policies is present in <system> as follows:

```
<qos-policies>
  <name>my-policy</name>
  <queue>
    <queue-id>1</queue-id>
    <maximum-burst-size>50</maximum-burst-size>
  </queue>
  <queue>
    <queue-id>2</queue-id>
    <maximum-burst-size>60</maximum-burst-size>
  </queue>
  <queue>
    <queue-id>3</queue-id>
    <maximum-burst-size>70</maximum-burst-size>
  </queue>
  <queue>
    <queue-id>4</queue-id>
    <maximum-burst-size>80</maximum-burst-size>
  </queue>
</qos-policies>
```

A client modifies the value of maximum-burst-size to 55 in queue-id 1:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <qos-policies>
        <name>my-policy</name>
        <queue>
          <queue-id>1</queue-id>
          <maximum-burst-size>55</maximum-burst-size>
        </queue>
      </qos-policies>
    </config>
  </edit-config>
</rpc>
```

Then the configuration of qos-policies is present in <operational> as follows:

```
        <qos-policies xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
                    or:origin="or:intended">
          <name>my-policy</name>
          <queue>
            <queue-id>1</queue-id>
            <maximum-burst-size>55</maximum-burst-size>
          </queue>
          <queue or:origin="or:system">
            <queue-id>2</queue-id>
            <maximum-burst-size>60</maximum-burst-size>
          </queue>
           <queue or:origin="or:system">
            <queue-id>3</queue-id>
            <maximum-burst-size>70</maximum-burst-size>
          </queue>
           <queue or:origin="or:system">
            <queue-id>4</queue-id>
            <maximum-burst-size>80</maximum-burst-size>
          </queue>
        </qos-policies>
```

4.4.2. Configuring Descendant Nodes of A System-defined Node

Suppose the system provides a loopback interface (named "lo0") with a default IPv4 address of "127.0.0.1" and a default IPv6 address of "::1".

The configuration of "lo0" interface is present in <system> as follows:

```
           <interfaces>
             <interface>
               <name>lo0</name>
               <ip-address>127.0.0.1</ip-address>
               <ip-address>::1</ip-address>
             </interface>
           </interfaces>
```

The configuration of "lo0" interface is present in <operational> as follows:

```
<interfaces xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
            or:origin="or:system">
  <interface>
    <name>lo0</name>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
  </interface>
</interfaces>
```

Later on, the client further configures the description node of a "lo0" interface as follows:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <interfaces>
        <interface>
          <name>lo0</name>
          <description>loopback</description>
        </interface>
      </interfaces>
    </config>
  </edit-config>
</rpc>
```

Then the configuration of interface "lo0" is present in <operational> as follows:

```
<interfaces xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
            or:origin="or:intended">
  <interface>
    <name>lo0</name>
    <description>loopback</description>
    <ip-address or:origin="or:system">127.0.0.1</ip-address>
    <ip-address or:origin="or:system">::1</ip-address>
  </interface>
</interfaces>
```

4.4.3. Declaring A System-defined Node in <running> Explicitly

In the environment which offline validation of <running> is required, a client need to declare the system-defined configurations that are actually referenced. Here is an example of a client explicitly declaring "lo0" in <running>. The client configures a "lo0" interface only with the list key "name" as follows:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <edit-config>
     <target>
       <running/>
     </target>
     <config>
       <interfaces>
         <interface>
           <name>lo0</name>
         </interface>
       </interfaces>
     </config>
   </edit-config>
 </rpc>
```

A read-back of <running> should looks like:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <edit-config>
     <target>
       <running/>
     </target>
     <config>
       <interfaces>
         <interface>
           <name>lo0</name>
         </interface>
       </interfaces>
     </config>
   </edit-config>
 </rpc>
```

5. Discovering System Configuration

There are two ways to discover system configuration: a "with-system" query parameter and a <system> configuration datastore.

5.1. The "with-system" Query Parameter

As defined in Section 1.3 and Section 1.4, All the system configuration will be implicitly hidden in <running>, hence the client can retrieve them through standard operations defined in YANG-driven management protocols such as NETCONF and RESTCONF with a "with-system" parameter to get a merged view.

All servers MUST implement a "with-system" parameter.

5.2. The <system> Configuration Datastore

This section is not applicable to non-NMDA servers. NMDA servers SHOULD implement a <system> configuration datastore, and they SHOULD also implement the <intended> datastore, which can be used as an alternative to "with-system" parameter.

Following guidelines for defining datastores in the appendix A of [RFC8342], this document introduces a new optional datastore resource named 'system' that represents the system configuration. A device MAY implement the mechanism defined in this document without implementing the "system" datastore, which would only eliminate the ability to programmatically determine the system configuration.

* Name: "system"

* YANG modules: all

* YANG nodes: all "config true" data nodes

* Management operations: The content of the datastore is set by the server in an implementation dependent manner. The content can not be changed by management operations via NETCONF, RESTCONF,the CLI, etc, but may change itself by upgrades and/or when resource-conditions are met. The datastore can be read using the standard NETCONF/RESTCONF protocol operations.

* Origin: This document does not define any new origin identity when it interacts with <intended> datastore and finally flows into <operational>. The "system" origin Metadata Annotation [RFC7952] is used to indicate the origin of a data item.

  Comment: Should we define any new origin identity to indicate new source of system configuration datastore?

* Protocols: YANG-driven management protocols, such as NETCONF and RESTCONF.

* Defining YANG module: "ietf-system".

The datastore's content is populated by the server and read-only to clients. Upon the content is created or changed, it will be merged into <intended> datastore. Unlike <factory-default>[RFC8808], it MAY change dynamically, e.g., depending on factors like during device upgrade or system-controlled resources(e.g., HW available) and the <system> datastore does not have to persist across reboots. <factory-reset> RPC operation defined in [RFC8088] can reset it to its factory default configuration without including configuration generated due to the system update or client-enabled functionality.

6. The "ietf-netconf-with-system" Module

6.1. Data Model Overview

This YANG module augments NETCONF <get> and <get-config> operation, which is designed to make implicitly hidden system configuration visible via a "with-system" parameter.

The following tree diagram [RFC8340] illustrates the "ietf-netconf-with-system" module:

```
module: ietf-netconf-with-system
  augment /nc:get-config/nc:input:
    +---w with-system?   empty
  augment /nc:get/nc:input:
    +---w with-system?   empty
```

The following tree diagram [RFC8340] illustrates "get" and "get-config" rpcs defined in "ietf-netconf" augmented by "ietf-netconf-with-system" module :

```
     rpcs:
       +---x get-config
       |  +---w input
       |  |  +---w source
       |  |  |  +---w (config-source)
       |  |  |     +--:(candidate)
       |  |  |     |  +---w candidate?   empty {candidate}?
       |  |  |     +--:(running)
       |  |  |     |  +---w running?     empty
       |  |  |     +--:(startup)
       |  |  |        +---w startup?     empty {startup}?
       |  |  +---w filter?        <anyxml>
       |  |  +---w with-system?   empty
       |  +--ro output
       |     +--ro data?   <anyxml>
       +---x get
       |  +---w input
       |  |  +---w filter?        <anyxml>
       |  |  +---w with-system?   empty
       |  +--ro output
       |     +--ro data?   <anyxml>
```

6.2. Example Usage

This section gives an example of request/response pairs with and without the "with-system" query parameter. The YANG module used are shown in Appendix C.2 of [RFC8342].

Suppose the following data is added to <running>:

```
{
    "bgp": {
        "local-as": "64501",
        "peer-as": "64502",
        "peer": {
            "name": "2001:db8::2:3"
        }
    }
}
```

All the messages are presented in a protocol-independent manner. JSON is used only for its conciseness.

REQUEST(without a "with-system" query parameter):

```
Target:/bgp
Query Parameter:
with-defaults: report-all
```

RESPONSE(both bgp/peer/local-as and bgp/peer/peer-as have default values for a peer. "local-port" leaf is not present in <running>):


```
{
    "bgp": {
        "local-as": "64501",
        "peer-as": "64502",
        "peer": {
            "name": "2001:db8::2:3",
            "local-as": "64501",
            "peer-as": "64502",
            "remote-port": "179",
            "state": "established"
        }
    }
}
```


REQUEST(with a "with-system" query parameter):

```
Target:/bgp
Query Parameter:
with-system
with-defaults: report-all
```

RESPONSE(local-port leaf value is supplied by the system):


```
{
    "bgp": {
        "local-as": "64501",
        "peer-as": "64502",
        "peer": {
            "name": "2001:db8::2:3",
            "local-as": "64501",
            "peer-as": "64502",
            "local-port": "60794",
            "remote-port": "179",
            "state": "established"
        }
    }
}
```


6.3. YANG Module

```
   <CODE BEGINS>
    file="ietf-netconf-with-system@2021-05-14.yang"
    module ietf-netconf-with-system {
       yang-version 1.1;
       namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-with-system";
       prefix ncws;

       import ietf-netconf {
         prefix nc;
         reference
           "RFC 6241: Network Configuration Protocol (NETCONF)";
       }

       organization
        "IETF NETMOD (Network Modeling) Working Group";

       contact
        "WG Web:   <http://tools.ietf.org/wg/netmod/>
         WG List:  <mailto:netmod@ietf.org>";
       description
        "This module defines an extension to the NETCONF protocol
         that allows the NETCONF client to control how system configuration
         data are handled by the server in particular NETCONF operations.

         Copyright (c) 2010 IETF Trust and the persons identified as
         the document authors.  All rights reserved.

         Redistribution and use in source and binary forms, with or
         without modification, is permitted pursuant to, and subject
         to the license terms contained in, the Simplified BSD License
         set forth in Section 4.c of the IETF Trust's Legal Provisions
         Relating to IETF Documents
         (http://trustee.ietf.org/license-info).

         This version of this YANG module is part of RFC XXXX; see
         the RFC itself for full legal notices.";
       // RFC Ed.: replace XXXX with actual RFC number and remove this note

       revision 2021-05-14 {
         description
           "Initial version.";
         reference
          "RFC XXXX: System configuration Data handling Behavior";
       }

     augment /nc:get-config/nc:input {
        description " Allows the get-config operation to use
          with-system to retrieve the complete system configuration.";
```

```
        leaf with-system {
            type empty ;
            description
              "Support system configuration retrieval on
               conventional configuration datastore. ";
           }
      }

     augment /nc:get/nc:input {
        description " Allows the get operation to use
          with-system to retrieve the complete system configuration.";
        leaf with-system {
            type empty ;
            description
              "Support system configuration retrieval on
               running datastore.";
            }
      }
    }
   <CODE ENDS>
```

7. The "ietf-system-datastore" Module

7.1. Data Model Overview

This YANG module defines a new YANG identity named "system" that uses the "ds:datastore" identity defined in [RFC8342]. Note that no new origin identity is defined in this document, the "or:system" origin Metadata Annotation [RFC7952] is used to indicate the origin of a data item.

The following diagram illustrates the relationship amongst the "identity" statements defined in the "ietf-system-datastore" and "ietf-datastores" YANG modules

```
Identities:
    +--- datastore
     |   +--- conventional
     |   |   +--- running
     |   |   +--- candidate
     |   |   +--- startup
     |   |   +--- system
     |   |   +--- intended
     |   +--- dynamic
     |   +--- operational
```

The diagram above uses syntax that is similar to but not defined in [RFC8340].

7.2. Example Usage

This section gives an example of data retrieval from <system>.

Suppose the following data is added to <running>:


```
{
    "bgp": {
        "local-as": "64501",
        "peer-as": "64502",
        "peer": {
            "name": "2001:db8::2:3"
        }
    }
}
```


All the messages are presented in a protocol-independent manner. JSON is used only for its conciseness.

REQUEST:

Datastore: <system>
Target:/bgp

RESPONSE("local-port" leaf value is supplied by the system):


```
{
    "bgp": {
        "peer": {
            "local-port": "60794"
        }
    }
}
```


7.3. YANG Module

```
<CODE BEGINS>
 file="ietf-system-datastore@2021-05-14.yang"
 module ietf-system-datastore {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-system-datastore";
    prefix sysds;

    import ietf-datastores {
      prefix ds;
      reference
        "RFC 8342: Network Management Datastore Architecture(NMDA)";
    }
```

```
     organization
      "IETF NETMDOD (Network Modeling) Working Group";

     contact
      "WG Web:   <http://tools.ietf.org/wg/netmod/>
       WG List:  <mailto:netmod@ietf.org>";
     description
      "This module defines a new YANG identity that uses the
       ds:datastore identity defined in [RFC8342].

       Copyright (c) 2010 IETF Trust and the persons identified as
       the document authors.  All rights reserved.

       Redistribution and use in source and binary forms, with or
       without modification, is permitted pursuant to, and subject
       to the license terms contained in, the Simplified BSD License
       set forth in Section 4.c of the IETF Trust's Legal Provisions
       Relating to IETF Documents
       (http://trustee.ietf.org/license-info).

       This version of this YANG module is part of RFC XXXX; see
       the RFC itself for full legal notices.";
     // RFC Ed.: replace XXXX with actual RFC number and remove this note

     revision 2021-05-14 {
       description

         "Initial version.";
       reference
        "RFC XXXX: System configuration Data handling Behavior";
     }

     identity system {
       base ds:conventional;
       description
         "This read-only datastore contains the complete configuration
          provided by the system itself.";
     }
 }
 <CODE ENDS>
```

8. IANA Considerations

8.1. The "IETF XML" Registry

This document registers two XML namespace URNs in the 'IETF XML registry', following the format defined in [RFC3688].

      URI: urn:ietf:params:xml:ns:yang:ietf-netconf-with-system
      Registrant Contact: The IESG.
      XML: N/A, the requested URIs are XML namespaces.

      URI: urn:ietf:params:xml:ns:yang:ietf-system-datastore
      Registrant Contact: The IESG.
      XML: N/A, the requested URIs are XML namespaces.

8.2. The "YANG Module Names" Registry

This document registers one module name in the 'YANG Module Names' registry, defined in [RFC6020] .

```
        name: ietf-netconf-with-system
        prefix: ncws
        namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-with-system
        RFC: XXXX // RFC Ed.: replace XXXX and remove this comment

        name: ietf-system-datastore
        prefix: sys
        namespace: urn:ietf:params:xml:ns:yang:ietf-system-datatstore
        RFC: XXXX // RFC Ed.: replace XXXX and remove this comment
```

9. Security Considerations

9.1. Regarding the "ietf-netconf-with-system" YANG Module

The YANG module defined in this document extends the base operations for NETCONF [RFC6241] and RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF users to a preconfigured subset of all available NETCONF protocol operations and content.

The security considerations for the base NETCONF protocol operations (see Section 9 of [RFC6241] apply to the new extended RPC operations defined in this document.

9.2. Regarding the "ietf-system-datastore" YANG Module

The YANG module defined in this document extends the base operations for NETCONF [RFC6241] and RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF users to a preconfigured subset of all available NETCONF protocol operations and content.

10. Contributors

Chongfeng Xie
China Telecom
Beijing
China

Email: xiechf@chinatelecom.cn

Kent Watsen
Watsen Networks

Email: kent+ietf@watsen.net

Jason Sterne
Nokia

Email: jason.sterne@nokia.com

Acknowledgements

Thanks to Robert Wilton, Balazs Lengyel, Andy Bierman, Jan Lindbland, Juergen Schoenwaelder, Alex Clemm, Timothy Carey for reviewing, and providing important input to, this document.

References

Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

Informative References

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <https://www.rfc-editor.org/info/rfc8407>.

[RFC8525] Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", RFC 8525, DOI 10.17487/RFC8525, March 2019, <https://www.rfc-editor.org/info/rfc8525>.

[RFC8808] Wu, Q., Lengyel, B., and Y. Niu, "A YANG Data Model for Factory Default Settings", RFC 8808, DOI 10.17487/RFC8808, August 2020, <https://www.rfc-editor.org/info/rfc8808>.

Appendix A. Key Use Cases

Following provides three use cases related to system-defined configuration lifecycle management. The simple interface data model defined in Appendix C.3 of [RFC8342] is used. For each use case, snippets of <running>, <system>, <intended> and <operational> are shown.

A.1. Device Powers On

<running>:

No configuration for lo0 appears in <running>;

<system>:

```
     <interfaces>
       <interface>
         <name>lo0</name>
         <ip-address>127.0.0.1</ip-address>
         <ip-address>::1</ip-address>
       </interface>
     </interfaces>
```

<intended>:

```
     <interfaces>
       <interface>
         <name>lo0</name>
         <ip-address>127.0.0.1</ip-address>
         <ip-address>::1</ip-address>
       </interface>
     </interfaces>
```

<operational>:

```
     <interfaces xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
                 or:origin="or:system">
       <interface>
         <name>lo0</name>
         <ip-address>127.0.0.1</ip-address>
         <ip-address>::1</ip-address>
       </interface>
     </interfaces>
```

A.2. Client Commits Configuration

If a client creates an interface "et-0/0/0" but the interface does not physically exist at this point:

<running>:

```
     <interfaces>
       <interface>
         <name>et-0/0/0</name>
         <description>Test interface</description>
       </interface>
     </interfaces>
```

<system>:

```
     <interfaces>
       <interface>
         <name>lo0</name>
         <ip-address>127.0.0.1</ip-address>
         <ip-address>::1</ip-address>
       </interface>
     </interfaces>
```

<intended>:

```
     <interfaces>
         <name>lo0</name>
         <ip-address>127.0.0.1</ip-address>
         <ip-address>::1</ip-address>
       </interface>
       <interface>
         <name>et-0/0/0</name>
         <description>Test interface</description>
       </interface>
       <interface>
     </interfaces>
```

<operational>:

```
     <interfaces xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
                 or:origin="or:intended">
       <interface or:origin="or:system">
         <name>lo0</name>
         <ip-address>127.0.0.1</ip-address>
         <ip-address>::1</ip-address>
       </interface>
     </interfaces>
```

A.3. Operator Installs Card into a Chassis

<running>:

```
     <interfaces>
       <interface>
         <name>et-0/0/0</name>
         <description>Test interface</description>
       </interface>
     </interfaces>
```

<system>:

```
        <interfaces>
          <interface>
            <name>lo0</name>
            <ip-address>127.0.0.1</ip-address>
            <ip-address>::1</ip-address>
          </interface>
          <interface>
            <name>et-0/0/0</name>
            <mtu>1500</mtu>
          </interface>
        </interfaces>
```

<intended>:

```
        <interfaces>
            <name>lo0</name>
            <ip-address>127.0.0.1</ip-address>
            <ip-address>::1</ip-address>
          </interface>
          <interface>
            <name>et-0/0/0</name>
            <description>Test interface</description>
            <mtu>1500</mtu>
          </interface>
          <interface>
        </interfaces>
```

<operational>:

```
        <interfaces xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
                    or:origin="or:intended">
          <interface or:origin="or:system">
            <name or:origin>lo0</name>
            <ip-address>127.0.0.1</ip-address>
            <ip-address>::1</ip-address>
          </interface>
         <interface>
            <name>et-0/0/0</name>
            <description>Test interface</description>
            <mtu or:origin="or:system">1500</mtu>
          </interface>
          <interface>
        </interfaces>
```

Appendix B. Changes between Revisions

v02 - v00

* Restructure the document content based on input in the system defined configuration interim meeting.

* Updates NMDA to define a read-only conventional configuration datastore called "system".

* Retrieval of implicit hidden system configuration via <get><get-config> with "with-system" parameter to support non-NMDA servers.

* Provide system defined configuration classification.

* Define Static Characteristics and dynamic behavior for system defined configuration.

* Separate "ietf-system-datastore" Module from "ietf-netconf-with-system" Module.

* Provide usage examples for dynamic behaviors.

* Provide usage examples for two YANG modules.

* Provide three use cases related to system-defined configuration lifecycle management.

* Classify the relation with <factory-default>.

Appendix C. Open Issues tracking

* Backward compatibility:consider the communication between the server and the new client or the old client simultaneously.

* Running always be valid? The client might need to understand how to merge if offline validation on running is used.

* Immutable flag

Authors' Addresses

Qiufang Ma (editor)
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China

Email: maqiufang1@huawei.com

Feng Chong
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China

Email: frank.fengchong@huawei.com

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China

Email: bill.wu@huawei.com