System-defined Configuration
draft-ietf-netmod-system-config-04

Abstract

This document describes how a management client and server handle YANG-modeled configuration data that is defined by the server itself. The system-defined configuration can be referenced (e.g. leafref) by configuration explicitly created by a client.

The Network Management Datastore Architecture (NMDA) defined in RFC 8342 is updated with a read-only conventional configuration datastore called "system" to hold system-defined configuration. As an alternative to clients explicitly copying referenced system-defined configuration into the target configuration datastore (e.g., <running>) so that the datastore is valid, a "resolve-system" parameter is defined to allow the server acting as a "system client" to copy referenced system-defined nodes automatically. This solution enables clients manipulating the target configuration datastore (e.g., <running>) to overlay (e.g., copy system configuration using the same key value as in <system>) and reference nodes defined in <system>, override values of configurations defined in <system>, and configure descendant nodes of system-defined nodes.

This document updates RFC 8342, RFC 6241, RFC 8526 and RFC 8040.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."
This Internet-Draft will expire on 8 May 2024.

Copyright Notice

Copyright (c) 2023 IETF Trust and the persons identified as the
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
license-info) in effect on the date of publication of this document.
Please review these documents carefully, as they describe your rights
and restrictions with respect to this document. Code Components
extracted from this document must include Revised BSD License text as
described in Section 4.e of the Trust Legal Provisions and are
provided without warranty as described in the Revised BSD License.

Table of Contents

1.  Introduction . . . . . . . . . . . . . . . . . . . . . . . .   3
  1.1.  Terminology . . . . . . . . . . . . . . . . . . . . . . .   5
  1.2.  Requirements Language . . . . . . . . . . . . . . . . . .  5
  1.3.  Updates to RFC 8342 . . . . . . . . . . . . . . . . . . .  6
  1.4.  Updates to RFC 6241 and RFC 8526 . . . . . . . . . . . .  6
  1.5.  Updates to RFC 8040 . . . . . . . . . . . . . . . . . . .  7
    1.5.1.  Query Parameter . . . . . . . . . . . . . . . . . . .  7
    1.5.2.  Query Parameter URI . . . . . . . . . . . . . . . . .  7
2.  Kinds of System Configuration . . . . . . . . . . . . . . . .   7
  2.1.  Immediately-Active . . . . . . . . . . . . . . . . . . .   8
  2.2.  Conditionally-Active . . . . . . . . . . . . . . . . . . .  8
  2.3.  Inactive-Until-Referenced . . . . . . . . . . . . . . . .  8
3.  The System Configuration Datastore (<system>) . . . . . . . .  8
4.  Static Characteristics of <system> . . . . . . . . . . . . . .  9
  4.1.  Read-only to Clients . . . . . . . . . . . . . . . . . . .  9
  4.2.  May Change via Software Upgrades or Resource Changes . . 10
  4.3.  No Impact to <operational> . . . . . . . . . . . . . . . . 10
5.  Dynamic Behavior . . . . . . . . . . . . . . . . . . . . . . . 10
  5.1.  Conceptual Model of Datastores . . . . . . . . . . . . . . 10
  5.2.  Explicit Declaration of System Configuration . . . . . . 13
  5.3.  Servers Auto-configuring Referenced System Configuration
       ("resolve-system" parameter) . . . . . . . . . . . . . . . . 14
  5.4.  Modifying (Overriding) System Configuration . . . . . . . 15
  5.5.  Examples . . . . . . . . . . . . . . . . . . . . . . . . . 16
    5.5.1.  Server Configuring of <running> Automatically . . . . 16
    5.5.2.  Declaring a System-defined Node in <running>
            Explicitly . . . . . . . . . . . . . . . . . . . . . . . . 21
    5.5.3.  Modifying a System-instantiated Leaf’s Value . . . . 24
    5.5.4.  Configuring Descendant Nodes of a System-defined
            Node . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
1. Introduction

The Network Management Datastore Architecture (NMDA) [RFC8342] defines system configuration as the configuration that is supplied by the device itself and appears in `<operational>` when it is in use (Figure 2 in [RFC8342]).

However, there is a desire to enable a server to better structure and expose the system configuration. NETCONF/RESTCONF clients can benefit from a standard mechanism to retrieve what system configuration is available on a server.
Some servers allow the NETCONF/RESTCONF client to reference a system-defined node which isn’t present in the target datastore (e.g., <running>). The absence of the system configuration in the datastore can render the datastore invalid from the perspective of a client or offline tools (e.g., missing leafref targets). This document describes several approaches to bring the datastore to a valid state and ensuring that all referential integrity constraints are satisfied.

Some servers allow the descendant nodes of system-defined configuration to be configured or modified. For example, the system configuration may contain an almost empty physical interface, while the client needs to be able to add, modify, or remove a number of descendant nodes. Some descendant nodes may not be modifiable (e.g., the interface "name" and "type" set by the system).

This document updates the Network Management Datastore Architecture (NMDA) defined in RFC 8342 with a read-only conventional configuration datastore called "system" to hold system-defined configuration. As an alternative to clients explicitly copying referenced system-defined configuration into the target configuration datastore (e.g., <running>) so that the datastore is valid, a "resolve-system" parameter is defined to allow the server acting as a "system client" to copy referenced system-defined nodes automatically. This solution enables clients manipulating the target configuration datastore (e.g., <running>) to overlay (e.g., copy system configuration using the same key value as in <system>) and reference nodes defined in <system>, override values of configurations defined in <system>, and configure descendant nodes of system-defined nodes.

If a system-defined node is referenced, it refers to one of the following cases throughout this document:

* It is present in a leafref "path" statement and referred as the leafref value
* It is used as an "instance-identifier" type value
* It is present in an Xpath expression of "when" or "must" constraints
* It is defined to satisfy the "mandatory" constraints
* It is defined to exactly satisfy the "min-element" constraints

Conformance to this document requires the NMDA servers to implement the "ietf-system-datastore" YANG module (Section 6).
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:

System configuration: Configuration that is provided by the system itself. System configuration is present in the system configuration datastore (regardless of being applied by the device or referenced by other configuration nodes), and appears in the intended configuration datastore. System configuration that is considered active (according to the NMDA defined in RFC 8342) appears in <operational> with origin="system". It is a different and separate concept from factory default configuration defined in RFC 8808 (which represents a preset initial configuration that is used to initialize the configuration of a server).

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

This document redefines the term "conventional configuration datastore" in Section 3 of [RFC8342] to add "system" to the list of conventional configuration datastores:

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.

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 8342

This document updates RFC 8342 to define a configuration datastore called "system" to hold system configuration, it also redefines the term "conventional configuration datastore" from RFC 8342 to add "system" to the list of conventional configuration datastores. The contents of <system> are read-only to clients but may change dynamically itself. <system> aware client may retrieve all three types of system configuration defined in Section 2, reference nodes defined in <system>, override values of configurations defined in <system>, and configure descendant nodes of system-defined nodes.

Configuration in <system> is merged with the current configuration of the device in <running> after the configuration transformations (e.g., template expansion, removal of inactive configuration defined in [RFC8342]) to create the contents of <intended>. As always, system configuration will appear in <operational> with origin="system" when it is in use.

The system datastore makes system configuration visible to clients in order for being referenced or configurable prior to present in <operational>.

1.4. Updates to RFC 6241 and RFC 8526

This document augments <edit-config> and <edit-data> RPC operations defined in [RFC6241] and [RFC8526] respectively, with a new additional input parameter "resolve-system". The <copy-config> RPC operation defined in [RFC6241] is also augmented to support "resolve-system" parameter.

The "resolve-system" parameter is optional and has no value. When it is provided and the server detects that there is a reference to a system-defined node during the validation, the server will automatically copy the referenced system configuration into the validated datastore to make the configuration valid without the client doing so explicitly. Legacy clients interacting with servers that support this parameter don’t see any changes in <edit-config>/<edit-data> and <copy-config> behaviors.

The server’s copy referenced nodes from <system> to the target datastore MUST be enforced at the end of the <edit-config>/<edit-data> or <copy-config> operations during the validation processing, regardless of which target datastore it is.

NETCONF server that supports this parameter MUST advertise the capability identifier:
1.5. Updates to RFC 8040

This document extends Sections 4.8 and 9.1.1 of [RFC8040] to add a new query parameter "resolve-system" and corresponding query parameter capability URI.

1.5.1. Query Parameter

The "resolve-system" parameter controls whether to allow a server copy any referenced system-defined configuration automatically without the client doing so explicitly. 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resolve-system</td>
<td>POST, PUT, PATCH</td>
<td>resolve any references not resolved by the client and copy referenced system configuration into &lt;running&gt; automatically. This parameter can be given in any order.</td>
</tr>
</tbody>
</table>

Figure 1: RESTCONF "resolve-system" Query Parameter

1.5.2. Query Parameter URI

To enable a RESTCONF client to discover if the "resolve-system" query parameter is supported by the server, the following capability URI is defined, which is advertised by the server if supported, using the "ietf-restconf-monitoring" module defined in RFC 8040:

urn:ietf:params:restconf:capability:resolve-system:1.0

2. Kinds of System Configuration

There are three types of system configurations defined in this document: immediately-active system configuration, conditionally-active system configuration, and inactive-until-referenced system configuration.
Active system configuration refers to configuration that is in use by a device. As per definition of the operational state datastore in [RFC8342], if system configuration is inactive, it should not appear in <operational>. However, system configuration is present in <system> once it is generated, regardless of whether it is active or not.

2.1. Immediately-Active

Immediately-active system configurations are those generated in <system> and applied immediately when the device is powered on (e.g., a loopback interface), irrespective of physical resource present or not, a special functionality enabled or not.

2.2. Conditionally-Active

System configurations which are generated in <system> and applied 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 a QoS feature is enabled, related QoS policies are automatically created by the system.

2.3. Inactive-Until-Referenced

There are some system configurations predefined (e.g., application ids, anti-x signatures, trust anchor certs, etc.) as a convenience for the clients, which must be referenced to be active. The clients can also define their own configurations for their unique requirements. Inactive-until-referenced system configurations are generated in <system> immediately when the device is powered on, but they are not active until being referenced.

3. The System Configuration Datastore (<system>)

NMUDA servers compliant with this document MUST implement a system configuration datastore, and they SHOULD also implement <intended>.

Following guidelines for defining datastores in the appendix A of [RFC8342], this document introduces a new datastore resource named ‘system’ that represents the system configuration.

* Name: "system"

* YANG modules: all
* YANG nodes: all "config true" data nodes up to the root of the tree, generated by the system

* 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 protocols such as NETCONF, RESTCONF, but may change itself by upgrades and/or when resource-conditions are met. The datastore can be read using the standard network management protocols such as NETCONF and RESTCONF.

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

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

* Defining YANG module: "ietf-system-datastore".

The datastore’s content is defined by the server and read-only to clients. Upon the content is created or changed, it will be merged into <intended>. Unlike <factory-default> [RFC8808], it MAY change dynamically, e.g., depending on factors like device upgrade or system-controlled resources change (e.g., HW available). The system configuration datastore doesn’t persist across reboots; the contents of <system> will be lost upon reboot and recreated by the system with the same or changed contents. <factory-reset> RPC operation defined in [RFC8808] can reset it to its factory default configuration without including configuration generated due to the system update or client-enabled functionality.

The system datastore is defined as a conventional configuration datastore and shares a common datastore schema with other conventional datastores.

4. Static Characteristics of <system>

4.1. Read-only to Clients

The system datastore is a read-only configuration datastore (i.e., edits towards <system> directly MUST be denied), though the client may be allowed to override the value of a system-initialized data node (see Section 5.4).
4.2. May Change via Software Upgrades or Resource Changes

System configuration may change dynamically, e.g., depending on factors like device upgrade or if system-controlled resources (e.g., HW available) change. In some implementations, when a QoS feature is enabled, QoS-related policies are created by the system.

If the system configuration gets changed, YANG notifications (e.g., "push-change-update" notification) [RFC6470][RFC8639][RFC8641] can be used to notify the client. Any update of the contents in <system> will not cause the automatic update of <running>, even if some of the system configuration has already been copied into <running> explicitly or automatically before the update.

4.3. No Impact to <operational>

This work intends to have no impact to <operational>. System configuration appears in <operational> with "origin=system". This document 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 out of scope, hence existing "config false" nodes are not impacted by this work.

5. Dynamic Behavior

5.1. Conceptual Model of Datastores

This document introduces a 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-active system configuration will be generated in <system> and active immediately, but inactive-until-referenced system configuration only becomes active if it is referenced by client-defined configuration. While conditionally-active system configuration will only be created and active if the condition on system resources is met when the device is powered on or running.

All above three types of system configurations will appear in <system>. Clients MAY reference nodes defined in <system>, override values of configurations defined in <system>, and configure descendant nodes of system-defined nodes, by copying or writing intended configurations into the target configuration datastore (e.g., <running>).
Configuration in <system> is merged with the current configuration of the device in <running> after the configuration transformations (e.g., template expansion, removal of inactive configuration defined in [RFC8342]) to create the contents of <intended>, in which process, the data node appearing in <running> takes precedence over the same node in <system> if the server allows the node to be modifiable; additional nodes to a list entry or new list/leaf-list entries appearing in <running> extends the list entry or the whole list/leaf-list defined in <system> if the server allows the list/leaf-list to be updated. In addition, the intended configuration datastore represents the configuration after all configuration transformation to <system> are performed (e.g., system-defined template expansion, removal of inactive system configuration). If a server implements <intended>, <system> MUST be merged into <intended>.

As a result, Figure 2 in Section 5 of RFC 8342 is updated with the below conceptual model of datastores which incorporates the system configuration datastore.
Servers MUST enforce that configuration references in <running> are resolved within <running> and ensure that <running> contains any referenced system configuration. Clients MUST either explicitly copy system-defined nodes into <running> or use the "resolve-system" parameter. The server MUST enforce that the referenced system nodes configured into <running> by the client is consistent with <system>. Note that <system> aware clients know how to discover what nodes exist in <system>. How clients unaware of the system datastore can find appropriate configurations is beyond the scope of this document.
No matter how the referenced system configurations are copied into <running>, the nodes copied into <running> would always be returned as the result of a read of <running>, regardless if the client is <system> aware.

Configuration defined in <system> is merged into <intended>. It is also present in <operational> if it is in use by the device, even if a client may delete the configuration which is copied from <system> into <running>. For example, system initializes a value for a particular leaf which is overridden by the client with a different value in <running>. The client may delete that node in <running>, in which case system-initialized value defined in <system> can be still in use and appear in <operational>.

Applied system configuration regardless of explicitly or automatically being copied into <running>, appears in <operational> with origin="system".

Comment: this might need further discussion: should the origin="system" be required for system configuration copied/pasted into <running>?

Any deletable system-provided configuration that is populated as part of <running> by the system at boot up, without being part of the contents of a <startup> datastore, must be defined in <factory-default> [RFC8808], which is used to initialize <running> when the device is first-time powered on or reset to its factory default condition.

5.2. Explicit Declaration of System Configuration

It is possible for a client to explicitly declare system configuration nodes in the target datastore (e.g., <running>) with the same values as in <system>, by configuring a node (list/leaf-list entry, leaf, etc.) in the target datastore (e.g., <running>) that matches the same node and value in <system>.

The explicit configuration of system-defined nodes in the target datastore (e.g., <running>) can be useful, for example, when the client doesn’t want a "system client" to have a role or hasn’t implemented the "resolve-system" parameter but need the datastore to be valid. The client can explicitly declare (i.e., configure in the datastore like <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 datastore appear valid.
5.3. Servers Auto-configuring Referenced System Configuration
("resolve-system" parameter)

This document defines a new parameter "resolve-system" to the input
for the <edit-config>, <edit-data>, and <copy-config> operations.
Clients that are aware of the "resolve-system" parameter MAY use this
parameter to avoid the requirement to provide a referentially
complete configuration in <running>.

If the "resolve-system" is present, and the server supports this
capability, the server MUST copy relevant referenced system-defined
nodes into the target datastore (e.g., <running>) without the client
doing the copy/paste explicitly, to resolve any references not
resolved by the client. The server acting as a "system client" like
any other remote clients copies the referenced system-defined nodes
when triggered by the "resolve-system" parameter.

The server may automatically configure the list entries (with at
least the keys) in the target datastore (e.g., <running>) for any
system configuration list entries that are referenced elsewhere by
the clients. Similarly, not all the contents of the list entry
(i.e., the descendant nodes) are necessarily copied by the server –
only the parts that are required to make <running> valid.

There is no distinction between the configuration in the target
datastore (e.g., <running>) which is automatically configured by the
server and the one explicitly declared by the client, e.g., a read
back of the datastore (i.e., <get>, <get-config> or <get-data>
operation) returns automatically configured nodes. Note that even an
auto-configured node is allowed to be deleted from the target
datastore by the client, the operation request (e.g., <edit-config>)
may not succeed due to incomplete referential integrity, it is also
possible that the system automatically configures the deleted node
again to make configuration valid, when a "resolve-system" parameter
is carried.

Once copied into <running>, system-instantiated node will not be
removed or updated automatically even all references to it are
deleted or system configuration no longer appears in <system> due to
factors like system-controlled resources change (e.g., HW
unavailable). That said, servers MAY upgrade system configuration as
well as any copied system nodes in <running> when licensing change or
device upgrade.

If the "resolve-system" parameter is not given by the client, the
server should not modify <running> in any way otherwise not specified
by the client. Not using capitalized "SHOULD NOT" in the previous
sentence is intentional. The intention is to bring awareness to the
general need to not surprise clients with unexpected changes. It is desirable for clients to always opt into using mechanisms having server-side changes. This document enables a client to opt into this behavior using the "resolve-system" parameter. An example of this type of opt-in behavior can also be found in RFC 7317, which enables a client to opt into its behavior using a "$0$" prefix (see ianach:crypt-hash type defined in [RFC7317]).

Support for the "resolve-system" parameter is OPTIONAL. Servers not supporting NMDA [RFC8342] MAY also implement this parameter without implementing the system configuration datastore, which would only eliminate the ability to expose the system configuration via protocol operations. If a server implements <system>, referenced system configuration is copied from <system> into the target datastore (e.g., <running>) when the "resolve-system" parameter is used; otherwise it is an implementation decision where to copy referenced system configuration into the target datastore (e.g., <running>).

Implementation specifics are beyond the scope of this document, however, due to the extra complexity brought by the "resolve-system" parameter, clients should be aware that it would cost a reasonable amount of time for the server to resolve reference, retrieve and copy the referenced system configuration from <system>, which could take multiple rounds since some errors may depend on the resolution of previous ones. Clients need to ensure that connections are alive during the server processing until a response is received.

5.4. Modifying (Overriding) System Configuration

In some cases, a server may allow some parts of system configuration to be modified. Modification of system configuration is achieved by the client writing configuration to <running> that overrides the system configuration. Configurations defined in <running> take precedence over system configuration nodes in <system> if the server allows the nodes to be modified.

For instance, descendant nodes in a system-defined list entry may be modifiable or not, even if some system configuration has been copied into <running> earlier. If a system node is non-modifiable, then writing a different value for that node MUST return an error. The immutability of system configuration is further defined in [I-D.ma-netmod-immutable-flag].

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).
5.5. Examples

This section shows some examples of server-configuring of <running> automatically, declaring a system-defined node in <running> explicitly, modifying a system-instantiated leaf’s value and configuring descendant nodes of a system-defined node. For each example, the corresponding XML snippets are provided.

5.5.1. Server Configuring of <running> Automatically

In this subsection, the following fictional module is used:

```yang
module example-application {
    yang-version 1.1;
    namespace "urn:example:application";
    prefix "app";

    import ietf-inet-types {
        prefix "inet";
    }

    container applications {
        list application {
            key "name";
            leaf name {
                type string;
            }
            leaf protocol {
                type enumeration {
                    enum tcp;
                    enum udp;
                }
            }
            leaf destination-port {
                type inet:port-number;
            }
        }
    }
}
```

The server may predefine some applications as a convenience for the clients. These predefined configurations are active only after being referenced by other configurations, which fall into the "inactive-until-referenced" system configuration as defined in Section 2. The system-instantiated application entries may be present in <system> as follows:
<applications xmlns="urn:example:application">
  <application>
    <name>ftp</name>
    <protocol>tcp</protocol>
    <destination-port>21</destination-port>
  </application>
  <application>
    <name>tftp</name>
    <protocol>udp</protocol>
    <destination-port>69</destination-port>
  </application>
  <application>
    <name>smtp</name>
    <protocol>tcp</protocol>
    <destination-port>25</destination-port>
  </application>
  ...
</applications>

The client may also define its customized applications. Suppose the configuration of applications is present in <running> as follows:

<applications xmlns="urn:example:application">
  <application>
    <name>my-app-1</name>
    <protocol>tcp</protocol>
    <destination-port>2345</destination-port>
  </application>
  <application>
    <name>my-app-2</name>
    <protocol>udp</protocol>
    <destination-port>69</destination-port>
  </application>
</applications>

A fictional ACL YANG module is used as follows, which defines a leafref for the leaf-list "application" data node to refer to an existing application name.
module example-acl {
  yang-version 1.1;
  namespace "urn:example:acl";
  prefix "acl";

  import example-application {
    prefix "app";
  }
  import ietf-inet-types {
    prefix "inet";
  }

  container acl {
    list acl_rule {
      key "name";
      leaf name {
        type string;
      }
      container matches {
        choice l3 {
          container ipv4 {
            leaf source_address {
              type inet:ipv4-prefix;
            }
            leaf dest_address {
              type inet:ipv4-prefix;
            }
          }
        }
        choice applications {
          leaf-list application {
            type leafref {
              path "/app:applications/app:application/app:name";
            }
          }
        }
      }
      leaf packet_action {
        type enumeration {
          enum forward;
          enum drop;
          enum redirect;
        }
      }
    }
  }
}
If a client configures an ACL rule referencing system predefined nodes which are not present in <running>, the client may issue an <edit-config> operation with the parameter "resolve-system" as follows:

```
<rpc message-id="101"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <edit-config>
   <target>
      <running/>
   </target>
   <config>
      <acl xmlns="urn:example:acl">
      <acl_rule>
         <name>allow_access_to_ftp_tftp</name>
         <matches>
            <ipv4>
               <source_address>198.51.100.0/24</source_address>
               <dest_address>192.0.2.0/24</dest_address>
            </ipv4>
            <application>ftp</application>
            <application>tftp</application>
            <application>my-app-1</application>
         </matches>
         <packet_action>forward</packet_action>
      </acl_rule>
      </acl>
   </config>
</edit-config>
```

Then following gives the configuration of applications in <running> which is returned in the response to a follow-up <get-config> operation:
<applications xmlns="urn:example:application">
  <application>
    <name>my-app-1</name>
    <protocol>tcp</protocol>
    <destination-port>2345</destination-port>
  </application>
  <application>
    <name>my-app-2</name>
    <protocol>udp</protocol>
    <destination-port>69</destination-port>
  </application>
  <application>
    <name>ftp</name>
  </application>
  <application>
    <name>tftp</name>
  </application>
</applications>

Then the configuration of applications is present in <operational> as follows:

<applications xmlns="urn:example:application"
               xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
               or:origin="or:intended">
  <application>
    <name>my-app-1</name>
    <protocol>tcp</protocol>
    <destination-port>2345</destination-port>
  </application>
  <application>
    <name>my-app-2</name>
    <protocol>udp</protocol>
    <destination-port>69</destination-port>
  </application>
  <application or:origin="or:system">
    <name>ftp</name>
    <protocol>tcp</protocol>
    <destination-port>21</destination-port>
  </application>
  <application or:origin="or:system">
    <name>tftp</name>
    <protocol>udp</protocol>
    <destination-port>69</destination-port>
  </application>
</applications>
Since the configuration of application "smtp" is not referenced by the client, and the server treats application "smtp" configuration as "inactive-until-referenced", it does not appear in <operational> but only in <system>.

5.5.2. Declaring a System-defined Node in <running> Explicitly

It’s also possible for a client to explicitly declare the system-defined configurations that are referenced. For instance, in the above example, the client MAY also explicitly configure the following system defined applications "ftp" and "tftp" only with the list key "name" before referencing:

```xml
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  
  <edit-config>  
    <target>  
      <running/>  
    </target>  
    <config>  
      <applications xmlns="urn:example:application">  
        <application>  
          <name>ftp</name>  
        </application>  
        <application>  
          <name>tftp</name>  
        </application>  
      </applications>  
    </config>  
  </edit-config>  
</rpc>
```

Then the client issues an <edit-config> operation to configure an ACL rule referencing applications "ftp" and "tftp" without the parameter "resolve-system" as follows:
Then following gives the configuration of applications in <running> which is returned in the response to a follow-up <get-config> operation, all the configuration of applications are explicitly configured by the client:
Then the configuration of applications is present in <operational> as follows:

```xml
<applications xmlns="urn:example:application"
               xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
               or:origin="or:intended">
  <application>
    <name>my-app-1</name>
    <protocol>tcp</protocol>
    <destination-port>2345</destination-port>
  </application>
  <application>
    <name>my-app-2</name>
    <protocol>udp</protocol>
    <destination-port>69</destination-port>
  </application>
  <application>
    <name>ftp</name>
    <protocol or:origin="or:system">tcp</protocol>
    <destination-port or:origin="or:system">21</destination-port>
  </application>
  <application>
    <name>tftp</name>
    <protocol or:origin="or:system">udp</protocol>
    <destination-port or:origin="or:system">69</destination-port>
  </application>
</applications>
```
Since the application names "ftp" and "tftp" are explicitly configured by the client, they take precedence over the values in <system>, the "origin" attribute will be set to "intended".

5.5.3. Modifying a System-instantiated Leaf’s Value

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

```yang
module example-qos-policy {
  yang-version 1.1;
  namespace "urn:example:qos";
  prefix "qos";

  container qos-policies {
    list policy {
      key "name";
      leaf name {
        type string;
      }
      list queue {
        key "queue-id";
        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 xmlns="urn:example:qos">
  <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 xmlns="urn:example:qos">
        <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="urn:example:qos"
  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>

5.5.4. Configuring Descendant Nodes of a System-defined Node

This subsection also uses the fictional interface YANG module defined in Appendix C.3 of [RFC8342]. 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:

```xml
<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>

6. The "ietf-system-datastore" Module
6.1. Data Model Overview

This YANG module defines a new YANG identity named "system" that uses the "ds:datastore" identity defined in [RFC8342]. A client can discover the system configuration datastore support on the server by reading the YANG library information from the operational state datastore. 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 is system. Support for the "origin" annotation is identified with the feature "origin" defined in [RFC8526].

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].

6.2. Example Usage

This section gives an example of data retrieval from <system>. The YANG module used are shown in Appendix C.2 of [RFC8342]. All the messages are presented in a protocol-independent manner. JSON is used only for its conciseness.

Suppose the following data is added to <running>:

```json
{
  "bgp": {
    "local-as": "64501",
    "peer-as": "64502",
    "peer": {
      "name": "2001:db8::2:3"
    }
  }
}
```
REQUEST (a <get-data> or GET request sent from the NETCONF or
RESTCONF client):

Datastore: <system>
Target:/bgp

An example of RESTCONF request:

GET /restconf/ds/system/bgp HTTP/1.1
Host: example.com
Accept: application/yang-data+xml

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

{
  "bgp": {
    "peer": {
      "name": "2001:db8::2:3",
      "local-port": "60794"
    }
  }
}

6.3. YANG Module

<CODE BEGINS> file "ietf-system-datastore@2023-11-05.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 NETDOD (Network Modeling) Working Group";
  contact
    "WG Web: https://datatracker.ietf.org/wg/netmod/
     WG List: NETMOD WG list <mailto:netmod@ietf.org>

    Author: Qiufang Ma
     <mailto:maqiufang1@huawei.com>
    Author: Qin Wu
     <mailto:bill.wu@huawei.com>
This module defines a new YANG identity that uses the ds: datastore identity defined in [RFC8342].

Copyright (c) 2023 IETF Trust and the persons identified as authors of the code. 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 Revised BSD License set forth in Section 4.c of the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info).

This version of this YANG module is part of RFC HHHH (https://www.rfc-editor.org/info/rfcHHHH); see the RFC itself for full legal notices.

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 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.

revision 2023-11-05 {
  description
    "Initial version.";
  reference
    "RFC XXXXX: System-defined Configuration";
}

identity system {
  base ds:conventional;
  description
    "This read-only datastore contains the configuration provided by the system itself.";
}

7. The "ietf-netconf-resolve-system" Module

This YANG module is optional to implement.
7.1. Data Model Overview

This YANG module augments NETCONF <edit-config>, <edit-data> and <copy-config> operations with a new parameter "resolve-system" in the input parameters. If the "resolve-system" parameter is present, the server will copy the referenced system configuration into target datastore automatically. A NETCONF client can discover the "resolve-system" parameter support on the server by checking the server’s capabilities included in the <hello> message.

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

```
module: ietf-netconf-resolve-system
  augment /nc:edit-config/nc:input:
    +---w resolve-system? empty
  augment /nc:copy-config/nc:input:
    +---w resolve-system? empty
  augment /ncds:edit-data/ncds:input:
    +---w resolve-system? empty
```

The following tree diagram [RFC8340] illustrates "edit-config", "copy-config" and "edit-data" rpcs defined in "ietf-netconf" and "ietf-netconf-nmda" respectively, augmented by "ietf-netconf-resolve-system" YANG module:

```
rpcs:
  +---x edit-config
    +---w input
      +---w target
        +---w (config-target)
          +---:(candidate)
            | +---w candidate? empty {candidate}?
          +---:(running)
            +---w running? empty {writable-running}?
        +---w default-operation? enumeration
        +---w test-option? enumeration {validate}?
        +---w error-option? enumeration
        +---w (edit-content)
          +---:(config)
            | +---w config? <anyxml>
          +---:(url)
            | +---w url? inet:uri {url}?
        +---w resolve-system? empty
  +---x copy-config
    +---w input
      +---w target
        +---w (config-target)
```
7.2. Example Usage

This section gives an example of an <edit-config> request to reference system-defined data nodes which are not present in <running> with a "resolve-system" parameter. A retrieval of <running> to show the auto-copied referenced system configurations after the <edit-config> request is also given. The YANG module used is shown as follows, leafrefs refer to an existing name and address of an interface:
module example-interface-management {
    yang-version 1.1;
    namespace "urn:example:interfacemgmt";
    prefix "inm";

    container interfaces {
        list interface {
            key name;
            leaf name {
                type string;
            }
            leaf description {
                type string;
            }
            leaf mtu {
                type uint16;
            }
            leaf ip-address {
                type inet:ip-address;
            }
        }
    }

    container default-address {
        leaf ifname {
            type leafref {
                path "../../interfaces/interface/name";
            }
        }
        leaf address {
            type leafref {
                path "../../interfaces/interface[name = current()]/..\ifname]" + "/ip-address";
            }
        }
    }
}

Image that the system provides a loopback interface (named "lo0") with a predefined MTU value of "1500" and a predefined IP address of "127.0.0.1", <system> shows the following configuration of loopback interface:
<interfaces xmlns="urn:example:interfacemgmt">
  <interface>
    <name>lo0</name>
    <mtu>1500</mtu>
    <ip-address>127.0.0.1</ip-address>
  </interface>
</interfaces>

The client sends an <edit-config> operation to add the configuration of default-address with a "resolve-system" parameter:

<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <default-address xmlns="urn:example:interfacemgmt">
        <if-name>lo0</if-name>
        <address>127.0.0.1</address>
      </default-address>
    </config>
    <resolve-system/>
  </edit-config>
</rpc>

Since the "resolve-system" parameter is provided, the server will resolve any leafrefs to system configurations and copy the referenced system-defined nodes into <running> automatically with the same value (i.e., the name and ip-address data nodes of lo0 interface) in <system> at the end of <edit-config> operation constraint enforcement. After the processing, a positive response is returned:

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

Then the client sends a <get-config> operation towards <running>: 

Ma, et al.               Expires 8 May 2024               [Page 34]
Given that the referenced interface "name" and "ip-address" of lo0 are configured by the server, the following response is returned:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <interfaces xmlns="urn:example:interfacemgmt">
      <interface>
        <name>lo0</name>
        <ip-address>127.0.0.1</ip-address>
      </interface>
    </interfaces>
  </data>
</rpc-reply>
```

7.3. YANG Module

```
<CODE BEGINS> file "ietf-netconf-resolve-system@2023-11-05.yang"

module ietf-netconf-resolve-system {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-netconf-resolve-system";
  prefix ncrs;

  import ietf-netconf {
    prefix nc;
    reference
      "RFC 6241: Network Configuration Protocol (NETCONF)";
  }
  import ietf-netconf-nmda {
    prefix ncds;
    reference
      "RFC 8526: NETCONF Extensions to Support the Network Management Datastore Architecture";
  }

```
organization
"IETF NETMOD (Network Modeling) Working Group";
contact
"WG Web:  <https://datatracker.ietf.org/wg/netmod/>
WG List:  <mailto:netmod@ietf.org>

Author: Qiufang Ma
<mailto:maqiufang1@huawei.com>
Author: Qin Wu
<mailto:bill.wu@huawei.com>
Author: Chong Feng
<mailto:fengchong1ly@gmail.com>";
description
"This module defines an extension to the NETCONF protocol
that allows the NETCONF client to control whether the server
is allowed to copy referenced system configuration
automatically without the client doing so explicitly.

Copyright (c) 2023 IETF Trust and the persons identified
as authors of the code. 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 Revised
BSD License set forth in Section 4.c of the IETF Trust’s
Legal Provisions Relating to IETF Documents

This version of this YANG module is part of RFC HHHH
(https://www.rfc-editor.org/info/rfcHHHH); see the RFC
itself for full legal notices.

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 (RFC 2119)
(RFC 8174) when, and only when, they appear in all
capitals, as shown here."

revision 2023-11-05 {
  description
    "Initial version.";
  reference
    "RFC XXXX: System-defined Configuration";
}

grouping resolve-system-grouping {
  description

"Define the resolve-system parameter grouping."
leaf resolve-system {
    type empty;
    description
        "When present, the server is allowed to automatically
         configure referenced system configuration into the
         target configuration datastore."
}

augment "/nc:edit-config/nc:input" {
    description
        "Allows the server to automatically configure
         referenced system configuration to make configuration
         valid."
    uses resolve-system-grouping;
}

augment "/nc:copy-config/nc:input" {
    description
        "Allows the server to automatically configure
         referenced system configuration to make configuration
         valid."
    uses resolve-system-grouping;
}

augment "/ncds:edit-data/ncds:input" {
    description
        "Allows the server to automatically configure
         referenced system configuration to make configuration
         valid."
    uses resolve-system-grouping;
}

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].
8.2. The "YANG Module Names" Registry

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

name: ietf-system-datastore
  prefix: sys
  maintained by IANA: N
  RFC: XXXX // RFC Ed.: replace XXXX and remove this comment

name: ietf-netconf-resolve-system
  prefix: ncrs
  maintained by IANA: N
  RFC: XXXX // RFC Ed.: replace XXXX and remove this comment

8.3. NETCONF Capability URN Registry

This document registers the following capability identifier URN in the 'Network Configuration Protocol (NETCONF) Capability URNs' registry:

urn:ietf:params:netconf:capability:resolve-system:1.0

8.4. RESTCONF Capability URN Registry

This document registers a capability in the 'RESTCONF Capability URNs' registry [RFC8040]:

<table>
<thead>
<tr>
<th>Index</th>
<th>Capability Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>:resolve-system</td>
<td>urn:ietf:params:restconf:capability:resolve-system:1.0</td>
</tr>
</tbody>
</table>

9. Security Considerations
9.1. 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.

9.2. Regarding the "ietf-netconf-resolve-system" YANG Module

The YANG module defined in this document extends the base operations for NETCONF [RFC6241] and [RFC8526]. 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.

10. Contributors
Acknowledgements

The authors would like to thank for following for discussions and providing input to this document (ordered by first name): Alex Clemm, Andy Bierman, Balazs Lengyel, Juergen Schoenwaelder, Martin Bjorklund, Mohamed Boucadair, Robert Wilton and Timothy Carey.

References

Normative References


Ma, et al. Expires 8 May 2024 [Page 40]
Informative References

[I-D.ma-netmod-immutable-flag]


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>:

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

<intended>:

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

<operational>:

```xml
<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>
    <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>
        <description>Test interface</description>
    </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>
        <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>
            <description>Test interface</description>
            <mtu>1500</mtu>
        </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>
    </interface>
</interfaces>

Appendix B. Changes between Revisions

v03 - v04
* Add some implementation consideration for "resolve-system" parameter
* Define a NETCONF capability identifier for "resolve-system" parameter so that the client can discover if it is supported by the server.
* state servers may upgrade copied system configuration in <running> as well during device upgrade or licensing change.

v02 - v03
* remove the merge mechanism related comments, as discussed in https://github.com/netconf-wg/netconf-next/issues/19
* Editorial changes

v01 - v02
* Define referenced system configuration
* better clarify "resolve-system" parameter
* update Figure 2 in NMDA RFC
* Editorial changes

v00 - v01
* Clarify why client’s explicit copy is not preferred but cannot be avoided if resolve-system parameter is not defined

* Clarify active system configuration

* Update the timing when the server’s auto copy should be enforced if a resolve-system parameter is used

* Editorial changes

Appendix C. Open Issues tracking

* Should the "with-origin" parameter be supported for <intended>?

Authors’ Addresses

Qiufang Ma (editor)
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China
Email: maqiufang1@huawei.com

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China
Email: bill.wu@huawei.com

Feng Chong
Email: fengchong1lly@gmail.com