Authors: Q. Ma, Ed.   K. Watsen   Q. Wu   C. Feng
         Huawei   Watsen Networks   Huawei   Huawei
         J. Lindblad
         Cisco Systems

System-defined Configuration

Abstract

This document updates NMDA [RFC8342] to define a read-only
conventional configuration datastore called "system" to hold system-
defined configurations. To avoid clients' explicit copy/paste of
referenced system-defined configuration into the target
configuration datastore (e.g., <running>), a "resolve-system"
parameter has been defined to allow the server acting as a "system
client" to copy referenced system-defined nodes automatically. The
solution enables clients manipulating the target configuration
datastore (e.g., <running>) to overlay and reference nodes defined
in <system>, override 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 [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, the client references a system configuration which
isn't present in the target datastore (e.g., <running>). Having to
copy the entire contents of the system configuration into the target
datastore should be avoided or reduced when possible while ensuring
that all referential integrity constraints are satisfied.

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).

This document updates NMDA [RFC8342] to define a read-only
conventional configuration datastore called "system" to hold system-
defined configurations. To avoid clients' explicit copy/paste of
referenced system-defined configuration into the target
configuration datastore (e.g., <running>), a "resolve-system"
parameter has been defined to allow the server acting as a "system
client" to copy referenced system-defined nodes automatically. The
solution enables clients manipulating the target configuration

datastore (e.g., <running>) to overlay and reference nodes defined in <system>, override values of configurations defined in <system>, and configure descendant nodes of system-defined nodes.

Conformance to this document requires servers to implement the "ietf-system-datastore" YANG Module.

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. System configuration is present in <system> once it's created (regardless of being applied by the device), and appears in <intended> which is subject to validation. Applied system configuration also appears in <operational> with origin="system".

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

This document redefines the term "conventional configuration datastore" from RFC 8342 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> datastore are read-only to clients but may
change dynamically. The <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.

The server will merge <running> and <system> to create <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, 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.

According to the NETCONF constraint enforcement model defined in the
section 8.3 of [RFC7950], if the target datastore of the <edit-
config>/<edit-data> or <copy-config> is "running" or "startup", 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. If the target
datastore of the <edit-config>/<edit-data> or <copy-config> is
"candidate", the server's copy referenced nodes from <system> to the
target datastore is delayed until a <commit> or <validate> operation
takes place.

1.5. Updates to RFC 8040

This document extends Section 4.8 and Section 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.

```
+----------------+---------+----------------------------------------+
| Name           | Methods | Description                            |
+----------------+---------+----------------------------------------+
|resolve-system  | POST,   | resolve any references not resolved by |
|                | PUT     | the client and copy referenced         |
|                |         | system configuration into <running>    |
|                |         | automatically. This parameter can be   |
|                |         | given in any order.                    |
+----------------+---------+----------------------------------------+
```

1.5.2. Query Parameter URI

To enable the 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: immediately-active
system configuration, conditionally-active system configuration and
inactive-until-referenced system configuration.

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 loop-back 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 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 generated in <system> immediately when it is powered on, but they are not applied and active until being referenced.

3. Static Characteristics

3.1. Read-only to Clients

The <system> configuration 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 4.4). Configuration defined in <system> is merged into <intended>, and present in <operational> if it is actively in use by the device. Thus unless the resource is no longer available (e.g., the interface removed physically), there is no way to actually delete system configuration from a server, even if a client may be allowed to delete the configuration copied from <system> into <running>. Any deletable system-provided configuration 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.

3.2. May Change via Software Upgrades

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 QoS function is enabled, QoS-related 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. 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.

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 out of scope, hence existing "config false" nodes are not impacted by this work.

4. Dynamic Behavior

4.1. Conceptual Model

This document introduces a mandatory 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 applied immediately but inactive-until-referenced system configuration only becomes active if it is referenced by client-defined configuration. While conditionally-active system configuration will be created and immediately applied 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>).

The server will merge <running> and <system> to create <intended>, in which process, the data node appears 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 appear 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>.

Servers MUST enforce that configuration references in <running> are resolved within the <running> datastore and ensure that <running> contains any referenced system objects. 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 objects are copied into
<running>, the nodes copied into <running> would always be returned
after a read of <running>, regardless if the client is <system>
aware.

4.2. 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>.

This explicit configuration of system-defined nodes in <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. 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.

4.3. Servers Auto-configuring Referenced System Configuration

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, 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. 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.

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 the <running> valid. A read

back of <running> (i.e., <get>, <get-config> or <get-data>
operation) returns those automatically copied nodes.

4.4. 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. Even if some system
configuration has been copied into <running> earlier, whether it is
modifiable or not in <running> follows general YANG and NACM rules,
and other server-internal restrictions. If a system configuration
node is non-modifiable, then writing a different value for that node
in <running> MUST return an error. The immutability of system
configuration is further defined in [I-D.ma-netmod-immutable-flag].

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.

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 unless the resource is no longer available. For
example, a "mandatory true" leaf 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. Another
example of this might be that system initializes a value for a
particular leaf which is overridden by the client with intended
value in <running>. The client may delete the leaf in <running>, but
system-initialized value defined in <system> will be in use and
appear in <operational>.

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.

4.5. Examples

This section shows the 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.

4.5.1. Server Configuring of <running> Automatically

In this subsection, the following fictional module is used:

```
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 objects are applied 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 destination_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>
              <destination_address>192.0.2.0/24</destination_add
            </ipv4>
            <application>ftp</application>
            <application>tftp</application>
            <application>my-app-1</application>
          </matches>
          <packet_action>forward</packet_action>
        </acl_rule>
      </acl>
    </config>
    <resolve-system/>
  </edit-config>
</rpc>
```

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, it does not appear in <operational> but only in
<system>.

4.5.2. 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:

```
<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:

```
<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>
              <destination_address>192.0.2.0/24</destination_add
            </ipv4>
            <application>ftp</application>
            <application>tftp</application>
            <application>my-app-1</application>
          </matches>
          <packet_action>forward</packet_action>
        </acl_rule>
      </acl>
    </config>
  </edit-config>
</rpc>
```

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:

```
<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>
    <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 as the value in
<system>, the "origin" attribute will be set to "intended".

4.5.3. Modifying a System-instantiated Leaf's Value

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

```
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>
```

4.5.4. 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:

```
    <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>
```

5. The <system> Configuration Datastore

NMDA servers claiming to support this document MUST implement a
<system> configuration datastore, and they SHOULD also implement the
<intended> datastore.

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. 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 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 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 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> datastore. 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> 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. The <system> configuration datastore must
always be valid, as defined in Section 8.1 of [RFC7950].

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> 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 [RF

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>:

```
{
    "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@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>
       Author: Qiufang Ma
               <mailto:maqiufang1@huawei.com>
       Author: Chong Feng
               <mailto:frank.fengchong@huawei.com>
       Author: Qin Wu
               <mailto:bill.wu@huawei.com>";

   description
    "This module defines a new YANG identity that uses the
     ds:datastore identity defined in [RFC8342].

     Copyright (c) 2021 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 Simplified
     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 2021-05-14 {
    description

      "Initial version.";
    reference
     "RFC XXXX: System-defined Configuration";
  }

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

<CODE ENDS>

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 YANG library
information with "ietf-netconf-resolve-system" included from the
operational state datastore.

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)
  |     |     +--:(candidate)
  |     |     |  +---w candidate?   empty {candidate}?
  |     |     +--:(running)
  |     |     |  +---w running?     empty {writable-running}?
  |     |     +--:(startup)
  |     |     |  +---w startup?     empty {startup}?
  |     |     +--:(url)
  |     |        +---w url?         inet:uri {url}?
  |     +---w source
  |     |  +---w (config-source)
  |     |     +--:(candidate)
  |     |     |  +---w candidate?   empty {candidate}?
  |     |     +--:(running)
  |     |     |  +---w running?     empty
  |     |     +--:(startup)
  |     |     |  +---w startup?     empty {startup}?
  |     |     +--:(url)
  |     |     |  +---w url?         inet:uri {url}?
  |     |     +--:(config)
  |     |        +---w config?      <anyxml>
  |     +---w resolve-system?      empty
  +---x edit-data
     +---w input
        +---w datastore            ds:datastore-ref
        +---w default-operation?   enumeration
        +---w (edit-content)
        |  +--:(config)
```

```
|  |  +---w config?         <anydata>
|  +--:(url)
|     +---w url?            inet:uri {nc:url}?
+---w resolve-system?       empty
```

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 objects 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". The <system> datastore 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 resonse 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>:

```
<rpc message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter type="subtree">
      <interfaces xmlns="urn:example:interfacemgmt"/>
    </filter>
  </get-config>
</rpc>
```

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@2021-05-14.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:   <http://tools.ietf.org/wg/netmod/>
      WG List:  <mailto:netmod@ietf.org>
      Author: Qiufang Ma
               <mailto:maqiufang1@huawei.com>
      Author: Chong Feng
               <mailto:frank.fengchong@huawei.com>
      Author: Qin Wu
               <mailto:bill.wu@huawei.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) 2021 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 Simplified
       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 2021-05-14 {
      description
        "Initial version.";
      reference
        "RFC XXXX: System-defined Configuration";
    }

  augment /nc:edit-config/nc:input {
    description
      "Allows the server to automatically configure
       referenced system configuration to make configuration
       valid.";
     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:copy-config/nc:input {
    description
      "Allows the server to automatically configure
       referenced system configuration to make configuration
       valid.";
     leaf resolve-system {
       type empty ;
       description
         "When present, the server is allowed to automatically
          configure referenced system configuration into the
          target configuration datastore.";
      }
   }

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

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

<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-system-datastore
Registrant Contact: The IESG.
XML: N/A, the requested URIs are XML namespaces.

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

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

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

8.3. RESTCONF Capability URN Registry

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

```
Index          Capability Identifier
-------------------------------------------------------------------
:resolve-system          urn:ietf:params:restconf:capability:resolve-
```

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

Chongfeng Xie
China Telecom
Beijing
China

Email: xiechf@chinatelecom.cn

Jason Sterne
Nokia

Email: jason.sterne@nokia.com

Acknowledgements

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

[I-D.ma-netmod-immutable-flag] Ma, Q., Wu, Q., and H. Li, "Immutable
           Metadata Annotation", Work in Progress, Internet-Draft,
           draft-ma-netmod-immutable-flag-00, 10 February 2022,
           <https://www.ietf.org/archive/id/draft-ma-netmod-
           immutable-flag-00.txt>.

[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 - v03

  *Define a RESTCONF capability URI for "resolve-system" RESTCONF
   query parameter;

  *Augment <copy-config> RPC operation to support "resolve-system"
   for input parameter;

  *Editorial changes for clarification and explanation. E.g.,
   definition of system configuration, is <system> always valid?
   Will the update of <system> be reflected into <running>? Clarify
   "read-only to clients" and "modifying system configuration", non-
   deletable system configuration, etc

v00 - v02

  *Remove the "with-system" parameter to retrieve <running> with
   system configuration merged in.

*Add a new parameter named "resolve-system" to allow the server to
    populate referenced system configuration into <running>
    automatically in order to make <running> valid.

   *Usage examples refinement.

  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

   *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

   Kent Watsen

Watsen Networks

Email: kent+ietf@watsen.net

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

Email: bill.wu@huawei.com

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

Email: frank.fengchong@huawei.com

Jan Lindblad
Cisco Systems

Email: jlindbla@cisco.com