NETMOD
Internet-Draft
Updates: 8342 (if approved)
Intended status: Standards Track
Expires: 17 July 2026

Q. Ma, Ed.
Q. Wu
Huawei
C. Feng

13 January 2026

System-defined Configuration
draft-ietf-netmod-system-config-18

Abstract

The Network Management Datastore Architecture (NMDA) in RFC 8342 defines several configuration datastores holding configuration. The contents of these configuration datastores are controlled by clients. This document introduces the concept of system configuration datastore holding configuration controlled by the system on which a server is running. The system configuration can be referenced (e.g., leafref) by configuration explicitly created by clients.

This document updates RFC 8342.

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 17 July 2026.

Table of Contents

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 expose the system configuration, regardless of whether it is in use. For example, some implementations define the system configuration which must be referenced to be active. NETCONF/RESTCONF clients can benefit from a standard mechanism to retrieve what system configuration is available on a server.

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, whose existence in the system configuration is tied to the presence of particular hardware, 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 "type" set by the system).

This document updates the NMDA defined in [RFC8342] with a read-only conventional configuration datastore called "system" to expose system-defined configuration. The solution enables configuration explicitly created by the clients to reference nodes defined in <system>, override system-provided values, and configure descendant nodes of system-defined configuration.

The solution defined in this document requires the use of NMDA for both clients and servers. Conformance to this document requires NMDA servers implement the "ietf-system-datastore" YANG module (Section 7).

1.1. Terminology

This document assumes that the reader is familiar with the contents of [RFC6241], [RFC7950], [RFC8342], and [RFC8525] and uses terminologies from those documents. The terms "device" and "server" are used interchangeably in this document.

The following terms are defined in this document:

system configuration: RFC 8342 defines it as "Configuration that is supplied by the device itself." The definition herein refines that definition of system configuration to represent configuration present in the system configuration datastore (regardless of whether it is applied or referenced). It may also be referred to as "system-defined configuration" or "system-provided configuration" throughout this document.

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. Note while protocol operations allow copying data between conventional datastores, the read-only nature of datastores such as <system> and <intended> restricts clients from copying data into them.

system node: An instance in the data tree that is provided by the system itself. System node may also be called "system-defined node" or "system-provided node" throughout this document.

referenced node: A referenced node is one of:

* Targets of leafref values defined via the "path" statement.

* Targets of "instance-identifier" type values.

* Nodes present in an XPath expression of "when" constraints.

* Nodes present in an XPath expression of "must" constraints.

* Nodes defined to satisfy the "mandatory true" constraints.

* Nodes defined to satisfy the "min-elements" constraints.

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 (Section 3), it also redefines the term "conventional configuration datastore" from [RFC8342] to add "system" to the list of conventional configuration datastores.

Configuration in <running> is merged with <system> to create the contents of <intended> after the configuration transformations (e.g., template expansion, removal of inactive configuration defined in [RFC8342]) have been performed, as described in Section 4.

This document also updates the definition of "intended" origin metadata annotation identity defined in Section 5.3.4 of [RFC8342]. The "intended" identity of origin value defined in [RFC8342] represents the origin of configuration provided by <intended>, this document updates its definition as the origin source of configuration explicitly provided by clients, and allows a subset of configuration in <intended> that flows from <system> yet is not configured or overridden explicitly in <running> to use "system" as its origin value. As per Section 5.3.4 of [RFC8342], all configuration with origin value being reported as "intended" MUST originate from <running>, which includes any configuration in <system> that has been copied into <running>. Configuration that is in <system> and not also present in <running> MUST be reported as origin "system" in <operational>.

2. Kinds of System Configuration

This document defines two types of system configuration. Configuration that is always-present and configuration that is conditionally-present. These types of system configuration are described in Section 2.1 and Section 2.2, respectively.

2.1. Always-Present

Always-present refers to system configuration which is generated in <system> when the device is powered on, irrespective of whether physical resources are present or whether a special functionality is enabled. An example of always-present system configuration is an always-existing loopback interface.

2.2. Conditionally-Present

Conditionally-present refers to system configuration which is generated in <system> based on specific conditions being met in a system. For example, if a physical resource is present (e.g., an interface card is inserted), the system automatically detects it and loads the associated configuration; when the physical resource is not present (an interface card is removed), the system configuration will automatically be removed from <system>. Another example is when a special functionality (e.g., a license or feature) is enabled, specific configuration may be created by the system.

3. The System Configuration Datastore (<system>)

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

* 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 datastore can be read using network management protocols such as NETCONF and RESTCONF, but its contents cannot be changed by management operations via NETCONF and RESTCONF protocols.

* Origin: This document does not define any new origin identity. The "system" identity of origin metadata annotation [RFC7952] is used to indicate the origin of a data item provided by the system.

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

* Defining YANG module: "ietf-system-datastore" (Section 7).

The system configuration datastore doesn’t persist across reboots.

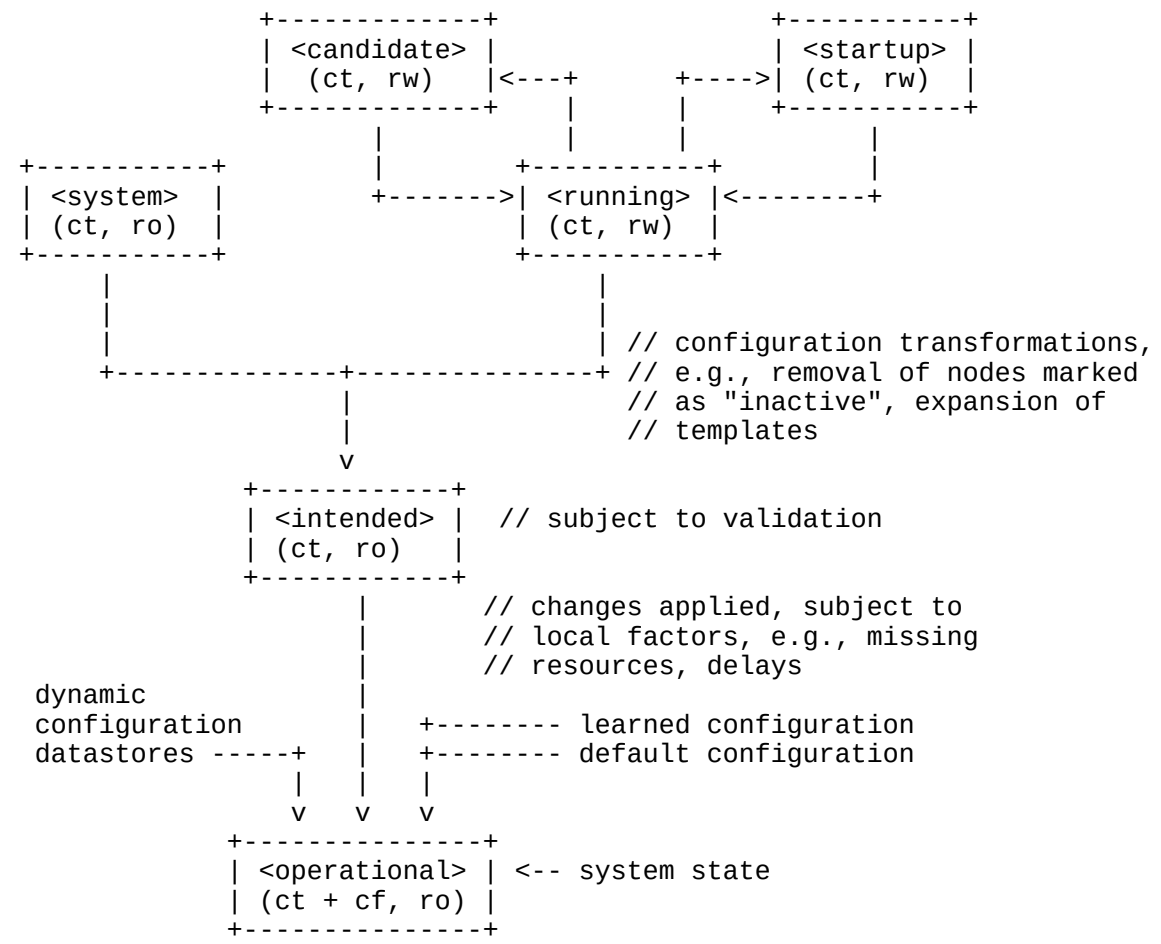
4. Conceptual Model of Datastores

Clients may provide configuration nodes that reference nodes defined in <system>, override system-provided values, and configure descendant nodes of system-defined configuration in <running>, as detailed in Section 6.

To ensure the validity of <intended>, configuration in <running> is merged with <system> to become <intended>, in which process, configuration appearing in <running> takes precedence over the same node in <system>. Since it is unspecified how to merge configuration before transformations, if <system> or <running> includes configuration that requires further transformation (e.g., template expansion, removal of inactive configuration defined in [RFC8342]) before it can be applied, configuration transformations MUST be performed independently on each datastore before <running> is merged with <system>.

Whenever configuration in <system> changes, the server MUST also immediately update and validate <intended>.

As a result, Figure 2 in Section 5 of [RFC8342] is updated with the below conceptual model of datastores which incorporates the system configuration datastore.



ct = config true; cf = config false
rw = read-write; ro = read-only
boxes denote named datastores

Figure 1: Architectural Model of Datastores

Configuration in <system> cannot be deleted by clients (e.g., a list entry can never be removed from <system> through protocol operations), even though a node defined in <system> may be overridden in <running>. If the system initializes a value for a particular leaf which is overridden by the client with a different value in <running> (Section 6.3), and if the node in <running> is removed at a later time, the system-initialized value defined in <system> appears in <intended> and may come into use eventually if applied successfully.

Configuration may disappear from <system> due to, e.g., resources no longer available. In such cases, configuration for missing resources can still remain in <running> and <intended>, but it will not be applied and appear in <operational>. This is further clarified in Section 5.3.2 of [RFC8342].

5. Static Characteristics

5.1. Read-only to Clients

The system datastore is read-only (i.e., edits towards <system> directly MUST be denied), though the client may be allowed to provide configuration that overrides the value of a system-initialized node (see Section 6.3).

5.2. No Changes to <operational>

This work does not change the definition of <operational>; it clarifies origin reporting, i.e., the origin of nodes sourced from <system> is reported as "system" unless explicitly configured or overridden in <running>. <system> enables 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.

6. Dynamic Behaviors

6.1. May Change via Software Upgrades or Resource Changes

The contents of <system> MAY change dynamically under various conditions, such as license change, software upgrade, and system-controlled resources change (see Section 2.2). The updates of system configuration may be obtained through YANG notifications (e.g., on-change notification) [RFC8639][RFC8641].

If system configuration changes (e.g., during a software upgrade), <running> SHOULD remain a valid configuration data tree. Any mechanisms to achieve this are outside the scope of this document.

6.2. Referencing System Configuration

Clients may create configuration data in <running> that references nodes in <system>. Some implementations may define system nodes solely as a convenience for clients to reference. It is also possible for the clients to define their customized nodes for reference.

Appendix A.1 provides an example of a client referencing system-defined nodes.

6.3. Modifying (Overriding) System Configuration

In some cases, a server may allow some parts of system configuration (e.g., a leaf's value) to be modified. Modification of system configuration is achieved by the client writing configuration data in <running> that overrides the values of matched configuration nodes at the corresponding level in <system>. Configurations defined in <running> take precedence over system configuration nodes in <system> if the server allows the nodes to be modified (some implementations may have immutable system configuration which is identified by the server using immutable metadata annotation, see [I-D.ietf-netmod-immutable-flag] for details).

Appendix A.2 provides an example of a client overriding a system-instantiated leaf's value.

6.4. Configuring Descendant nodes of System Configuration

A server may also allow a client to add 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).

Appendix A.3 provides an example of a client configuring descendant nodes of a system-defined node.

7. The "ietf-system-datastore" Module

7.1. Data Model Overview

This YANG module defines a new YANG identity named "system" that uses the "ds:conventional" identity defined in [RFC8342] as its base. A client can discover the system configuration datastore support on the server by reading the YANG library information from the operational state datastore.

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

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

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

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

7.2. YANG Module

<CODE BEGINS> file "ietf-system-datastore@2026-01-14.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 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:fengchongllly@gmail.com>";
  description
    "This module defines a new YANG identity that uses the
     ds:conventional identity defined in [RFC8342].

     Copyright (c) 2026 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 XXXX
        (https://www.rfc-editor.org/info/rfcXXXX); see the RFC
        itself for full legal notices.";

    revision 2026-01-14 {
      description
        "Initial version.";
      reference
        "RFC XXXX: System-defined Configuration";
    }

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

  <CODE ENDS>
```

7.3. Example Usage

The following example shows how the configuration in <system> could be retrieved in a NETCONF <get-data> RPC operation. The example uses the "example-application" fictional data model defined in Appendix A.1.

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-nmda"
     xmlns:sysds="urn:ietf:params:xml:ns:yang:ietf-system-datastore">
    <datastore>sysds:system</datastore>
    <subtree-filter>
      <applications xmlns="urn:example:application"/>
    </subtree-filter>
  </get-data>
</rpc>
```

When using the RESTCONF protocol, the system configuration datastore can be accessed via the resource: {+restconf}/ds/ietf-system-datastore:system. The following example uses an HTTP GET method to request "applications" configuration:

```
GET /restconf/ds/ietf-system-datastore:system/\
   example-application:applications HTTP/1.1
Host: example.com
Accept: application/yang-data+xml
```

8. IANA Considerations

8.1. The "IETF XML" Registry

This document registers one XML namespace URI 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.
```

8.2. The "YANG Module Names" Registry

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

```
      name: ietf-system-datastore
      prefix: sysds
      namespace: urn:ietf:params:xml:ns:yang:ietf-system-datastore
      maintained by IANA? N
      RFC: XXXX // RFC Ed.: replace XXXX and remove this comment
```

9. Operational Considerations

System configuration exists regardless of whether the server implements <system> or not. The introduction of <system> provides a standardized way to expose system configuration within NMDA.

NMDA clients that are not aware of <system> will continue to operate correctly. They will interact only with datastores such as <running>, <candidate>, <intended>, and <operational> as before. The presence of <system> does not change the fundamental behavior for such legacy clients. Operators should be aware that to fully leverage the capabilities defined in this document, client applications need to be updated to recognize and interact with system configuration.

10. Security Considerations

10.1. Considerations for the "ietf-system-datastore" YANG Module

This section is modeled after the template described in Section 3.7 of [I-D.ietf-netmod-rfc8407bis].

The "ietf-system-datastore" YANG module defines a data model that is designed to be accessed via YANG-based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. These protocols have to use a secure transport layer (e.g., SSH [RFC4252], TLS [I-D.ietf-tls-rfc8446bis], and QUIC [RFC9000]) and have to use mutual authentication.

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

The YANG module only defines an identity that uses the "ds:conventional" identity as its base. The module by itself does not expose any data nodes that are writable, data nodes that contain read-only state, or RPCs. As such, there are no additional security issues related to the YANG module that need to be considered.

10.2. Considerations for System Configuration

The system datastore, while read-only to clients, may contain sensitive information such as hardware identifiers, security policies, and critical system resources. Read access to sensitive system nodes and subtrees within the datastore MUST be controlled to prevent unauthorized disclosure. Implementations are strongly advised to log all access attempts to sensitive system configuration for audit purposes.

Furthermore, while <system> cannot be modified directly, system configuration may be overridden as a merging result (Section 6.3). An attacker may configure a leaf that shadows a sensitive node in <system>. Misconfiguration in <running> could lead to unintended system behavior including security policy bypass and availability risks. Unauthorized modification to sensitive contents MUST be prevented to avoid those negative effects on the network.

11. References

11.1. 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>.

[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>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[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>.

[RFC8639] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Subscription to YANG Notifications", RFC 8639, DOI 10.17487/RFC8639, September 2019, <https://www.rfc-editor.org/info/rfc8639>.

[RFC8641] Clemm, A. and E. Voit, "Subscription to YANG Notifications for Datastore Updates", RFC 8641, DOI 10.17487/RFC8641, September 2019, <https://www.rfc-editor.org/info/rfc8641>.

11.2. Informative References

[I-D.ietf-netmod-immutable-flag] Ma, Q., Wu, Q., Lengyel, B., and H. Li, "YANG Metadata Annotation for Immutable Flag", Work in Progress, Internet-Draft, draft-ietf-netmod-immutable-flag-06, 27 November 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-immutable-flag-06>.

[I-D.ietf-netmod-rfc8407bis] Bierman, A., Boucadair, M., and Q. Wu, "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", Work in Progress, Internet-Draft, draft-ietf-netmod-rfc8407bis-28, 5 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-rfc8407bis-28>.

[I-D.ietf-tls-rfc8446bis] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-

ietf-tls-rfc8446bis-14, 13 September 2025,
<https://datatracker.ietf.org/doc/html/draft-ietf-tls-rfc8446bis-14>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, DOI 10.17487/RFC4252, January 2006, <https://www.rfc-editor.org/info/rfc4252>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

[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>.

[RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <https://www.rfc-editor.org/info/rfc7952>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[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>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[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>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

Appendix A. Example of Dynamic Behaviors

This section presents some sample data models and corresponding contents of various datastores with different dynamic behaviors described in Section 6. The XML snippets are used only for illustration purposes. Note this section does not show the contents of <intended> as they are related to the configuration in <operational> assuming the intended configuration is applied successfully. Also note that if the "origin" metadata annotation for configuration is unspecified in snippets, it is inherited from its parent node.

A.1. Referencing System-defined Nodes

In this subsection, the following fictional module is used:

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

  import ietf-inet-types {
    prefix "inet";
  }
  container applications {
    list application {
      key "name";
      leaf name {
        type string;
      }
      leaf app-id {
        type string;
      }
      leaf protocol {
        type enumeration {
          enum tcp;
          enum udp;
        }
        mandatory true;
      }
      leaf destination-port {
        default "0";
        type inet:port-number;
      }
      leaf description {
        type string;
      }
      container security-protection {
        presence "Indicates that security protection is enabled.";
        leaf risk-level {
          type enumeration {
            enum high;
            enum low;
          }
        }
        //additional leafs for security-specific configuration...
      }
    }
  }
}
```

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 "ex-acl";

  import example-application {
    prefix "ex-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 src-address {
              type inet:ipv4-prefix;
            }
            leaf dst-address {
              type inet:ipv4-prefix;
            }
          }
        }
        choice applications {
          leaf-list application {
            type leafref {
              path "/ex-app:applications/ex-app:application"
                 + "/ex-app:name";
            }
          }
        }
      }
      leaf packet-action {
        type enumeration {
          enum forward;
          enum drop;
          enum redirect;
        }
      }
    }
  }
```

```
}
```

The server may predefine some applications as a convenience for clients, these applications are immediately-present system configuration. When the device is powered on, the system-instantiated application entries may be present in <system> as follows:

```
<applications xmlns="urn:example:application">
  <application>
    <name>ftp</name>
    <app-id>001</app-id>
    <protocol>tcp</protocol>
    <destination-port>21</destination-port>
    <security-protection>
      <risk-level>low</risk-level>
    </security-protection>
  </application>
  <application>
    <name>tftp</name>
    <app-id>002</app-id>
    <protocol>udp</protocol>
    <destination-port>69</destination-port>
    <security-protection>
      <risk-level>low</risk-level>
    </security-protection>
  </application>
  <application>
    <name>smtp</name>
    <app-id>003</app-id>
    <protocol>tcp</protocol>
    <destination-port>25</destination-port>
    <security-protection>
      <risk-level>low</risk-level>
    </security-protection>
  </application>
</applications>
```

The client may also define customized applications. Those applications may be present in <running> as follows:

```
<applications xmlns="urn:example:application">
  <application>
    <name>my-smtp</name>
    <app-id>101</app-id>
    <protocol>tcp</protocol>
    <destination-port>2345</destination-port>
    <description>customized smtp application</description>
    <security-protection>
      <risk-level>high</risk-level>
    </security-protection>
  </application>
  <application>
    <name>my-foo</name>
    <app-id>102</app-id>
    <protocol>udp</protocol>
    <destination-port>1024</destination-port>
    <description>customized application</description>
  </application>
</applications>
```

If a client configures an ACL rule referencing some system-provided or customized applications, the configuration of ACL rule may be shown as follows:

```
<acl xmlns="urn:example:acl">
  <acl-rule>
    <name>allow-access-to-ftp-tftp</name>
    <matches>
      <ipv4>
        <src-address>198.51.100.0/24</src-address>
        <dst-address>192.0.2.0/24</dst-address>
      </ipv4>
      <application>ftp</application>
      <application>tftp</application>
      <application>my-smtp</application>
    </matches>
    <packet-action>forward</packet-action>
  </acl-rule>
</acl>
```

As different entries of application configuration in <system> and <running> are merged to create <intended>, and there are no merging conflicts in the contents between <system> and <running>, <operational> might contain the configuration of applications with the values of origin reflecting the source of entries as follows:

```
   <applications xmlns="urn:example:application"
                 xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
                 or:origin="or:intended">
     <application>
       <name>my-smtp</name>
       <app-id>101</app-id>
       <protocol>tcp</protocol>
       <destination-port>2345</destination-port>
       <description>customized smtp application</description>
       <security-protection>
         <risk-level>high</risk-level>
       </security-protection>
     </application>
     <application>
       <name>my-foo</name>
       <app-id>102</app-id>
       <protocol>udp</protocol>
       <destination-port>1024</destination-port>
       <description>customized application</description>
     </application>
     <application or:origin="or:system">
       <name>ftp</name>
       <app-id>001</app-id>
       <protocol>tcp</protocol>
       <destination-port>21</destination-port>
       <security-protection>
         <risk-level>low</risk-level>
       </security-protection>
     </application>
     <application or:origin="or:system">
       <name>tftp</name>
       <app-id>002</app-id>
       <protocol>udp</protocol>
       <destination-port>69</destination-port>
       <security-protection>
         <risk-level>low</risk-level>
       </security-protection>
     </application>
     <application or:origin="or:system">
       <name>smtp</name>
       <app-id>003</app-id>
       <protocol>tcp</protocol>
       <destination-port>25</destination-port>
       <security-protection>
         <risk-level>low</risk-level>
       </security-protection>
     </application>
   </applications>
```

A.2. Modifying a System-instantiated Leaf's Value

This subsection uses the following fictional interface YANG module:

```
module example-interface {
  yang-version 1.1;
  namespace "urn:example:interface";
  prefix "ex-if";

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

  container interfaces {
    list interface {
      key name;
      leaf name {
        type string;
      }
      leaf description {
        type string;
      }
      leaf mtu {
        type uint32;
      }
      leaf-list ip-address {
        type inet:ip-address;
      }
    }
  }
}
```

Suppose the system provides an always-present loopback interface (named "lo0") with an MTU value "65536", a default IPv4 address of "127.0.0.1", and a default IPv6 address of "::1". The configuration of "lo0" interface may be present in <system> as follows:

```
<interfaces xmlns="urn:example:interface">
  <interface>
    <name>lo0</name>
    <mtu>65536</mtu>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
  </interface>
</interfaces>
```

A client modifies the value of MTU to 9216 by adding the following configuration into <running>:

```
<interfaces xmlns="urn:example:interface">
  <interface>
    <name>lo0</name>
    <mtu>9216</mtu>
  </interface>
</interfaces>
```

Since the MTU value provided by the client takes precedence over the system-provided value, and the "origin" value of configuration provided by the client is set to "intended", the configuration of interfaces that is present in <operational> may be as follows:

```
<interfaces xmlns="urn:example:interface"
            xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
            or:origin="or:intended">
  <interface>
    <name>lo0</name>
    <mtu>9216</mtu>
    <ip-address or:origin="or:system">127.0.0.1</ip-address>
    <ip-address or:origin="or:system">::1</ip-address>
  </interface>
</interfaces>
```

A.3. Configuring Descendant Nodes of a System-defined Node

Based on the example in Appendix A.2, imagine the client further adds the description node of a "lo0" interface in <running> as follows:

```
<interfaces xmlns="urn:example:interface">
  <interface>
    <name>lo0</name>
    <description>loopback</description>
  </interface>
</interfaces>
```

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

```
   <interfaces xmlns="urn:example:interface"
               xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
               or:origin="or:intended">
     <interface>
       <name>lo0</name>
       <description>loopback</description>
       <mtu>9216</mtu>
       <ip-address or:origin="or:system">127.0.0.1</ip-address>
       <ip-address or:origin="or:system">::1</ip-address>
     </interface>
   </interfaces>
```

Appendix B. Key Use Cases

This section provides three use cases related to how <system> interacts with other datastores (e.g., <candidate>, <running>, <intended>, and <operational>). The following fictional interface data model is used:

```
module example-interface-management {
  yang-version 1.1;
  namespace "urn:example:interfacemgmt";
  prefix "ex-ifm";

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

  container interfaces {
    list interface {
      key "name";
      leaf name {
        type string;
      }
      leaf type {
        type enumeration {
          enum ethernet;
          enum atm;
          enum loopback;
        }
      }
      leaf enabled {
        type boolean;
        default "true";
      }
      leaf-list ip-address {
        type inet:ip-address;
      }
      leaf speed {
        when "../type = 'ethernet'";
        type enumeration {
          enum 10Mb;
          enum 100Mb;
        }
      }
      leaf description {
        type string;
      }
    }
  }
}
```

For each use case, corresponding sample configuration in <running>, <system>, <intended> and <operational> are shown. The XML snippets are used only for illustration purposes.

B.1. Device Powers On

When the device is powered on, suppose the system provides an always-present loopback interface (named "lo0") which is not explicitly configured in <running>. Thus, no configuration for interfaces appears in <running>;

And the contents of <system> are:

```
<interfaces xmlns="urn:example:interfacemgmt">
  <interface>
    <name>lo0</name>
    <type>loopback</type>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
    <description>system-defined interface</description>
  </interface>
</interfaces>
```

In this case, the configuration of loopback interface is only present in <system>, the configuration of interface in <intended> would be identical to the one in <system> shown above.

And <operational> will show the system-provided loopback interface, note that <operational> also includes the default value specified in the YANG module:

```
<interfaces xmlns="urn:example:interfacemgmt"
            xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
            or:origin="or:system">
  <interface>
    <name>lo0</name>
    <type>loopback</type>
    <enabled or:origin="or:default">true</enabled>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
    <description>system-defined interface</description>
  </interface>
</interfaces>
```

B.2. Client Commits Configuration

If a client creates an interface "et-0/0/0" but the interface does not physically exist at this point, what is in <running> appears as follows:

```
<interfaces xmlns="urn:example:interfacemgmt">
  <interface>
    <name>et-0/0/0</name>
    <ip-address>192.168.10.10</ip-address>
    <description>pre-provisioned interface</description>
  </interface>
</interfaces>
```

And the contents of <system> remain unchanged, only containing the "lo0" loopback interface, since the interface "et-0/0/0" is not physically present:

```
<interfaces xmlns="urn:example:interfacemgmt">
  <interface>
    <name>lo0</name>
    <type>loopback</type>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
    <description>system-defined interface</description>
  </interface>
</interfaces>
```

The contents of <intended> represent the merged data of <system> and <running>:

```
<interfaces xmlns="urn:example:interfacemgmt">
  <interface>
    <name>lo0</name>
    <type>loopback</type>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
    <description>system-defined interface</description>
  </interface>
  <interface>
    <name>et-0/0/0</name>
    <ip-address>192.168.10.10</ip-address>
    <description>pre-provisioned interface</description>
  </interface>
</interfaces>
```

Since the interface named "et-0/0/0" does not exist, the associated configuration is not present in <operational>, which appears as follows:

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

B.3. Operator Installs Card into a Chassis

When the interface is installed by the operator, the system will detect it and generate the associated conditionally-present interface configuration in <system>. The contents of <running> keep unchanged:

```
<interfaces xmlns="urn:example:interfacemgmt">
  <interface>
    <name>et-0/0/0</name>
    <ip-address>192.168.10.10</ip-address>
    <description>pre-provisioned interface</description>
  </interface>
</interfaces>
```

And <system> might appear as follows:

```
<interfaces xmlns="urn:example:interfacemgmt">
  <interface>
    <name>lo0</name>
    <type>loopback</type>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
    <description>system-defined interface</description>
  </interface>
  <interface>
    <name>et-0/0/0</name>
    <type>ethernet</type>
    <description>system-defined interface</description>
  </interface>
</interfaces>
```

Then <intended> contains the merged configuration of <system> and <running>:

```
<interfaces xmlns="urn:example:interfacemgmt">
  <interface>
    <name>lo0</name>
    <type>loopback</type>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
    <description>system-defined interface</description>
  </interface>
  <interface>
    <name>et-0/0/0</name>
    <type>ethernet</type>
    <ip-address>192.168.10.10</ip-address>
    <description>pre-provisioned interface</description>
  </interface>
</interfaces>
```

And the contents of <operational> appear as follows:

```
<interfaces xmlns="urn:example:interfacemgmt"
            xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
            or:origin="or:intended">
  <interface or:origin="or:system">
    <name>lo0</name>
    <type>loopback</type>
    <enabled or:origin="or:default">true</enabled>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
    <description>system-defined interface</description>
  </interface>
  <interface>
    <name>et-0/0/0</name>
    <type or:origin="or:system">ethernet</type>
    <enabled or:origin="or:default">true</enabled>
    <ip-address>192.168.10.10</ip-address>
    <description>pre-provisioned interface</description>
  </interface>
</interfaces>
```

B.4. Client further Commits Configuration

If the client further sets the speed of interface "et-0/0/0" in <running>:

```
<interfaces xmlns="urn:example:interfacemgmt">
  <interface>
    <name>et-0/0/0</name>
    <speed>10Mb</speed>
  </interface>
</interfaces>
```

The contents of <system> keep unchanged:

```
<interfaces xmlns="urn:example:interfacemgmt">
  <interface>
    <name>lo0</name>
    <type>loopback</type>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
    <description>system-defined interface</description>
  </interface>
  <interface>
    <name>et-0/0/0</name>
    <type>ethernet</type>
    <description>system-defined interface</description>
  </interface>
</interfaces>
```

And the contents of <intended> which represents a merged results of <running> and <system> are as follows:

```
<interfaces xmlns="urn:example:interfacemgmt">
  <interface>
    <name>lo0</name>
    <type>loopback</type>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
    <description>system-defined interface</description>
  </interface>
  <interface>
    <name>et-0/0/0</name>
    <type>ethernet</type>
    <ip-address>192.168.10.10</ip-address>
    <speed>10Mb</speed>
    <description>pre-provisioned interface</description>
  </interface>
</interfaces>
```

And <operational> would appear as follows:

```
   <interfaces xmlns="urn:example:interfacemgmt"
               xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
               or:origin="or:intended">
     <interface or:origin="or:system">
       <name>lo0</name>
       <type>loopback</type>
       <enabled>true</enabled>
       <ip-address>127.0.0.1</ip-address>
       <ip-address>::1</ip-address>
       <description>system-defined interface</description>
     </interface>
     <interface>
       <name>et-0/0/0</name>
       <type or:origin="or:system">ethernet</type>
       <enabled or:origin="or:default">true</enabled>
       <ip-address>192.168.10.10</ip-address>
       <speed>10Mb</speed>
       <description>pre-provisioned interface</description>
     </interface>
   </interfaces>
```

Acknowledgements

The authors would like to thank for following for discussions and providing input to this document: Balazs Lengyel, Robert Wilton, Juergen Schoenwaelder, Andy Bierman, Martin Bjorklund, Mohamed Boucadair, Michal Va邸ko, Alexander Clemm, and Timothy Carey.

Contributors

Kent Watsen
Watsen Networks
Email: kent+ietf@watsen.net

Jan Lindblad
Cisco Systems
Email: jlindbla@cisco.com

Jason Sterne
Nokia
Email: jason.sterne@nokia.com

Chongfeng Xie
China Telecom
Beijing
China
Email: xiechf@chinatelecom.cn

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

Chong Feng
Email: fengchonglly@gmail.com