Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2017

M. Bjorklund, Ed.
Tail-f Systems
J. Schoenwaelder
Jacobs University
P. Shafer
K. Watsen
Juniper
R. Wilton
Cisco

October 27, 2016

A Revised Conceptual Model for YANG Datastores
draft-nmdsdt-netmod-revised-datastores-00

Abstract

Datastores are a fundamental concept binding the YANG data modeling language to protocols transporting data defined in YANG data models, such as NETCONF or RESTCONF. This document defines a revised conceptual model of datastores based on the experience gained with the initial simpler model and addressing requirements that were not well supported in the initial model.

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 http://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 April 30, 2017.

Table of Contents

1. Introduction

This document provides a revised architectural framework for datastores as they are used by network management protocols such as

NETCONF [RFC6241], RESTCONF [I-D.ietf-netconf-restconf] and the YANG [RFC7950] data modeling language. Datastores are a fundamental concept binding management data models to network management protocols and agreement on a common architectural model of datastores ensures that data models can be written in a network management

protocol agnostic way. This architectural framework identifies a set of conceptual datastores but it does not mandate that all network management protocols expose all these conceptual datastores. Furthermore, the architecture does not detail how data is encoded by network management protocols.

2. Background

NETCONF [RFC6241] provides the following definitions:

- datastore: A conceptual place to store and access information. A datastore might be implemented, for example, using files, a database, flash memory locations, or combinations thereof.

- configuration datastore: The datastore holding the complete set of configuration data that is required to get a device from its initial default state into a desired operational state.

YANG 1.1 [RFC7950] provides the following refinements when NETCONF is used with YANG (which is the usual case but note that NETCONF was defined before YANG did exist):

- datastore: When modeled with YANG, a datastore is realized as an instantiated data tree.

- configuration datastore: When modeled with YANG, a configuration datastore is realized as an instantiated data tree with configuration data.

RFC 6244 defined operational state data as follows:

- Operational state data is a set of data that has been obtained by the system at runtime and influences the system's behavior similar to configuration data. In contrast to configuration data, operational state is transient and modified by interactions with internal components or other systems via specialized protocols.

Section 4.3.3 of RFC 6244 discusses operational state and among other things mentions the option to consider operational state as being stored in another datastore. Section 4.4 of this document then concludes that at the time of the writing, modeling state as a separate data tree is the recommended approach.

Implementation experience and requests from operators [I-D.ietf-netmod-opstate-reqs], [I-D.openconfig-netmod-opstate] indicate that the datastore model initially designed for NETCONF and refined by YANG needs to be extended. In particular, the notion of intended configuration and applied configuration has developed.

Furthermore, separating operational state data from configuration data in a separate branch in the data model has been found operationally complicated. The relationship between the branches is not machine readable and filter expressions operating on configuration data and on related operational state data are different.

3. Terminology

This document defines the following terms:

- configuration data: Data that determines how a device behaves. Configuration data can originate from different sources. In YANG 1.1, configuration data is the "config true" nodes.

- static configuration data: Configuration data that is eventually persistent and used to get a device from its initial default state into its desired operational state.

- dynamic configuration data: Configuration data that is obtained dynamically during the operation of a device through interaction with other systems and not persistent.

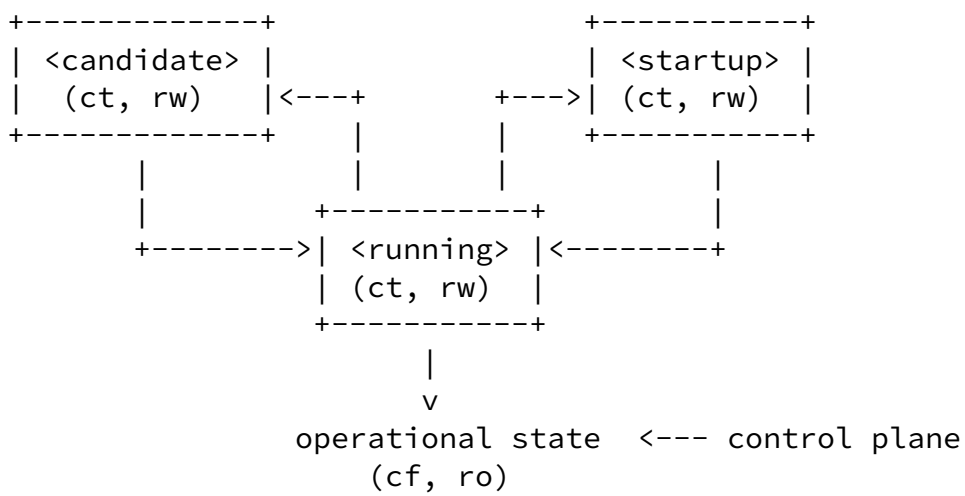
- system configuration data: Configuration data that is supplied by the device itself.

- data-model-defined configuration data: Configuration data that is not explicitly provided but for which a value defined in the data model is used. In YANG 1.1, such data can be defined with the

"default" statement or in "description" statements.

4. Original Model of Datastores

The following drawing shows the original model of datastores as it is currently used by NETCONF [RFC6241]:



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

Note that read-only (ro) and read-write (rw) is to be understood at a conceptual level. In NETCONF, for example, support for the <candidate> and <startup> datastores is optional and the <running> datastore does not have to be writable. Furthermore, the <startup>

datastore can only be modified by copying <running> to <startup> in the standardized NETCONF datastore editing model. The RESTCONF protocol does not expose these differences and instead provides only a writable unified datastore, which hides whether edits are done through a <candidate> datastore or by directly modifying the <running> datastore or via some other implementation specific mechanism. RESTCONF also hides how configuration is made persistent. Note that implementations may also have additional datastores that can propagate changes to the <running> datastore. NETCONF explicitly mentions so called named datastores.

Some observations:

- Operational state has not been defined as a datastore although there were proposals in the past to introduce an operational state datastore.

- The NETCONF <get/> operation returns the content of the <running> configuration datastore together with the operational state. It is therefore necessary that config false data is in a different branch than the config true data if the operational state data can have a different lifetime compared to configuration data or if configuration data is not immediately or successfully applied.

- Several implementations have proprietary mechanisms that allow clients to store inactive data in the <running> datastore; this

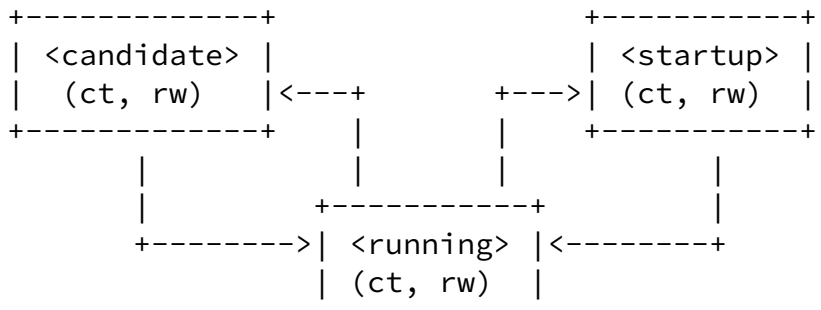
  inactive data is only exposed to clients that indicate that they support the concept of inactive data; clients not indicating support for inactive data receive the content of the <running> datastore with the inactive data removed. Inactive data is conceptually removed during validation.

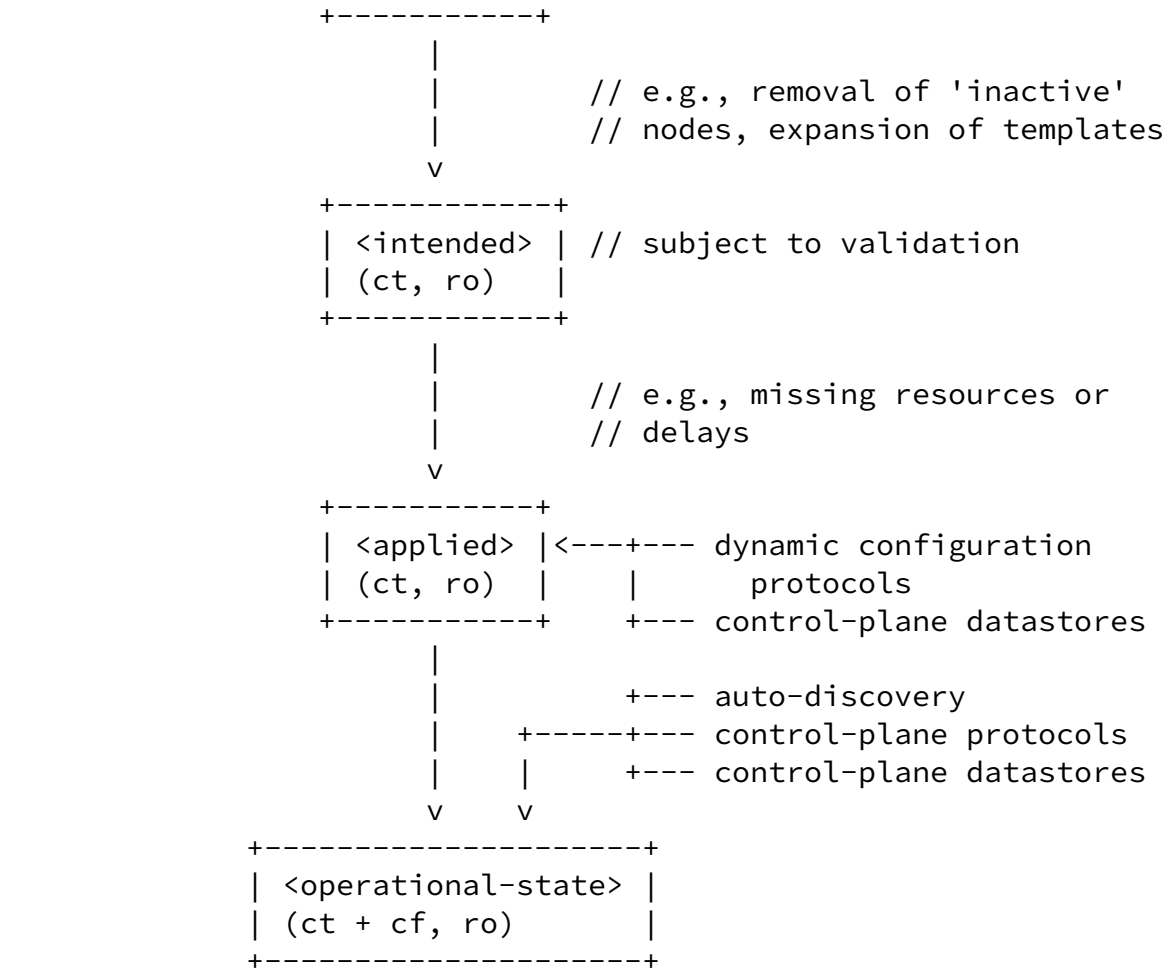
- Some implementations have proprietary mechanisms that allow clients to define configuration templates in <running>. These templates are expanded automatically by the system, and the resulting configuration is applied internally.

- Some operators have reported that it is essential for them to be able to retrieve the configuration that has actually been successfully applied, which may be a subset or a superset of the <running> configuration.

5. Revised Model of Datastores

Below is a new conceptual model of datastores extending the original model in order reflect the experience gained with the original model.





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

The model foresees control-plane datastores that are by definition not part of the persistent configuration of a device. In some contexts, these have been termed ephemeral datastores since the information is ephemeral, i.e., lost upon reboot. The control-plane datastores interact with the rest of the system through the <applied> or <operational-state> datastores, depending on the type of data they contain. Note that the ephemeral datastore discussed in I2RS documents maps to a control-plane datastore in the revised datastore model described here.

5.1. The <intended> datastore

The <intended> datastore is a read-only datastore that consists of config true nodes. It is tightly coupled to <running>. When data is written to <running>, the data that is to be validated is also conceptually written to <intended>. Validation is performed on the contents of <intended>.

On a traditional NETCONF implementation, <running> and <intended> are always the same.

Currently there are no standard mechanisms defined that affect <intended> so that it would have different contents than <running>, but this architecture allows for such mechanisms to be defined.

One example of such a mechanism is support for marking nodes as inactive in <running>. Inactive nodes are not copied to <intended>, and are thus not taken into account when validating the configuration.

Another example is support for templates. Templates are expanded when copied into <intended>, and the result is validated.

5.2. The <applied> datastore

The <applied> datastore is a read-only datastore that consists of config true nodes. It contains the currently active configuration on the device. This data can come from several sources; from <intended>, from dynamic configuration protocols (e.g., DHCP), or from control-plane datastores.

As data flows into the <applied> and <operational-state> datastores, it is conceptually marked with a metadata annotation ([RFC7952]) that indicates its origin. The "origin" metadata annotation is defined in Section 8. The values are YANG identities. The following identities are defined:

```
  +-- origin
      +-- static
      +-- dynamic
      +-- data-model
      +-- system
```

These identities can be further refined, e.g., there might be an identity "dhcp" derived from "dynamic".

The <applied> datastore contains the subset of the instances in the <operational-state> datastore where the "origin" values are derived from or equal to "static" or "dynamic".

5.2.1. Missing Resources

Sometimes some parts of <intended> configuration refer to resources that are not present and hence parts of the <intended> configuration cannot be applied. A typical example is an interface configuration that refers to an interface that is not currently present. In such a situation, the interface configuration remains in <intended> but the interface configuration will not appear in <applied>.

5.2.2. System-controlled Resources

Sometimes resources are controlled by the device and such system controlled resources appear in (and disappear from) the <operational-state> dynamically. If a system controlled resource has matching configuration in <intended> when it appears, the system will try to apply the configuration, which causes the configuration to appear in <applied> eventually (if application of the configuration was successful).

5.3. The <operational-state> datastore

The <operational-state> datastore is a read-only datastore that consists of config true and config false nodes. In the original NETCONF model the operational state only had config false nodes. The reason for incorporating config true nodes here is to be able to expose all operational settings without having to replicate definitions in the data models.

The <operational-state> datastore contains all configura data actually used by the system, i.e., all applied configuration, system configuration and data-model-defined configuration. This data is marked with the "origin" metadata annotation. In addition, the <operational-state> datastore also contains state data.

In the <operational-state> datastore, semantic constraints defined in the data model are not applied. See Appendix B.

6. Implications

6.1. Implications on NETCONF

- A mechanism is needed to announce support for <intended>,

- Support for <intended>, <applied>, and <operational-state> should be optional to implement.

- For systems supporting <intended> or <applied> configuration datastores, the <get-config/> operation may be used to retrieve data stored in these new datastores.

- A new operation should be added to retrieve the operational state data store (e.g., <get-state/>). An alternative is to define a new operation to retrieve data from any datastore (e.g., <get-data> with the name of the datastore as a parameter). In principle <get-config/> could work but it would be a confusing name.

- The <get/> operation will be deprecated since it returns data from two datastores that may overlap in the revised datastore model.

6.1.1. Migration Path

A common approach in current data models is to have two separate trees "/foo" and "/foo-state", where the former contains config true nodes, and the latter config false nodes. A data model that is designed for the revised architectural framework presented in this document will have a single tree "/foo" with a combination of config true and config false nodes.

A server that implements the <operational-state> datastore can implement a module of the old design. In this case, some instances are probably reported both in the "/foo" tree and in the "/foo-state" tree.

A server that does not implement the <operational-state> datastore can implement a module of the new design, but with limited functionality. Specifically, it may not be possible to retrieve all operationally used instances (e.g., dynamically configured or system-controlled). The same limitation applies to a client that does not implement the <operational-state> datastore, but talks to a server that implements it.

6.2. Implications on RESTCONF

- The {+restconf}/data resource represents the combined configuration and state data resources that can be accessed by a client. This is effectively bundling <running> together with <operational-state>, much like the <get/> operation of NETCONF. This design should be deprecated.

- A new query parameter is needed to indicate that data from <operational-state> is requested.

6.3. Implications on YANG

- Some clarifications may be needed if this revised model is adopted. YANG currently describes validation in terms of the <running> configuration datastore while it really happens on the <intended> configuration datastore.

6.4. Implications on Data Models

- Since the NETCONF <get/> operation returns the content of the <running> configuration datastore and the operational state together in one tree, data models were often forced to branch at the top-level into a config true branch and a structurally similar config false branch that replicated some of the config true nodes and added state nodes. With the revised datastore model this is not needed anymore since the different datastores handle the different lifetimes of data objects. Introducing this model together with the deprecation of the <get/> operation makes it possible to write simpler models.

- There may be some differences in the value set of some nodes that are used for both configuration and state. At this point of time, these are considered to be rare cases that can be dealt with using different nodes for the configured and state values.

- It is important to design data models with clear semantics that work equally well for instantiation in a configuration datastore and instantiation in the <operational-state> datastore.

7. Data Model Design Guidelines

7.1. Auto-configured or Auto-negotiated Values

Sometimes configuration leafs support special values that instruct the system to automatically configure a value. An example is an MTU that is configured to 'auto' to let the system determine a suitable MTU value. Another example is Ethernet auto-negotiation of link speed. In such a situation, it is recommended to model this as two separate leafs, one config true leaf for the input to the auto-negotiation process, and one config false leaf for the output from the process.

8. Data Model

```
<CODE BEGINS> file "ietf-yang-architecture@2016-10-13.yang"

module ietf-yang-architecture {
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-architecture";
  prefix arch;

  import ietf-yang-metadata {
    prefix md;
  }

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

  contact
    "WG Web:   <https://datatracker.ietf.org/wg/netmod/>

     WG List:  <mailto:netmod@ietf.org>

     Editor:   Martin Bjorklund
               <mailto:mbj@tail-f.com>";

  description
    "This YANG module defines an 'origin' metadata annotation,
```

```
      and a set of identities for the origin value.  The 'origin'
      metadata annotation is used to mark data in the applied
      and operational state datastores with information on where
      the data originated.

      Copyright (c) 2016 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
      (http://trustee.ietf.org/license-info).

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

    revision 2016-10-13 {
      description
        "Initial revision.";
      reference
```



```
        "RFC XXXX: A Revised Conceptual Model for YANG Datastores";
    }

    /*
     * Identities
     */

    identity origin {
      description
        "Abstract base identitiy for the origin annotation.";
    }

    identity static {
      base origin;
      description
        "Denotes data from static configuration (e.g., <intended>).";
    }
```

```
  identity dynamic {
    base origin;
    description
      "Denotes data from dynamic configuration protocols
       or dynamic datastores (e.g., DHCP).";
  }

  identity system {
    base origin;
    description
      "Denotes data created by the system independently of what
       has been configured.";
  }

  identity data-model {
    base origin;
    description
      "Denotes data that does not have an explicitly configured
       value, but has a default value in use.  Covers both simple
       defaults and complex defaults.";
  }

  /*
   * Metadata annotations
   */

  md:annotation origin {
    type identityref {
      base origin;
    }
```



```
  }

}
```

<CODE ENDS>

9. IANA Considerations

TBD

10. Security Considerations

This document discusses a conceptual model of datastores for network management using NETCONF/RESTCONF and YANG. It has no security impact on the Internet.

11. Acknowledgments

This document grew out of many discussions that took place since 2010. Several Internet-Drafts ([I-D.bjorklund-netmod-operational], [I-D.wilton-netmod-opstate-yang], [I-D.ietf-netmod-opstate-reqs], [I-D.kwatsen-netmod-opstate], [I-D.openconfig-netmod-opstate]) and [RFC6244] touched on some of the problems of the original datastore model. The following people were authors to these Internet-Drafts or otherwise actively involved in the discussions that led to this document:

- Lou Berger, LabN Consulting, L.L.C., <lberger@labn.net>
- Andy Bierman, YumaWorks, <andy@yumaworks.com>
- Marcus Hines, Google, <hines@google.com>
- Christian Hopps, Deutsche Telekom, <chopps@chopps.org>
- Acee Lindem, Cisco Systems, <acee@cisco.com>
- Ladislav Lhotka, CZ.NIC, <lhotka@nic.cz>
- Thomas Nadeau, Brocade Networks, <tnadeau@lucidvision.com>
- Anees Shaikh, Google, <aashaikh@google.com>
- Rob Shakir, Google, <robjs@google.com>

Juergen Schoenwaelder was partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

12. References

12.1. Normative References

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-17 (work in progress), September 2016.

[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, <http://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, <http://www.rfc-editor.org/info/rfc7950>.

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

12.2. Informative References

[I-D.bjorklund-netmod-operational]
Bjorklund, M. and L. Lhotka, "Operational Data in NETCONF and YANG", draft-bjorklund-netmod-operational-00 (work in progress), October 2012.

[I-D.ietf-netmod-opstate-reqs]
Watsen, K. and T. Nadeau, "Terminology and Requirements for Enhanced Handling of Operational State", draft-ietf-netmod-opstate-reqs-04 (work in progress), January 2016.

[I-D.kwatsen-netmod-opstate]
Watsen, K., Bierman, A., Bjorklund, M., and J. Schoenwaelder, "Operational State Enhancements for YANG, NETCONF, and RESTCONF", draft-kwatsen-netmod-opstate-02 (work in progress), February 2016.

[I-D.openconfig-netmod-opstate]
Shakir, R., Shaikh, A., and M. Hines, "Consistent Modeling of Operational State Data in YANG", draft-openconfig-netmod-opstate-01 (work in progress), July 2015.

[I-D.wilton-netmod-opstate-yang]
Wilton, R., ""With-config-state" Capability for NETCONF/RESTCONF", draft-wilton-netmod-opstate-yang-02 (work in progress), December 2015.

[RFC6244] Shafer, P., "An Architecture for Network Management Using NETCONF and YANG", RFC 6244, DOI 10.17487/RFC6244, June 2011, <http://www.rfc-editor.org/info/rfc6244>.

Appendix A. Example Data

In this example, the following fictional module is used:

```
module example-system {
  yang-version 1.1;
  namespace urn:example:system;
  prefix sys;

  import ietf-inet-types {
    prefix inet;
  }

  container system {
    leaf hostname {
      type string;
    }

    list interface {
      key name;

      leaf name {
        type string;
      }

      container auto-negotiation {
        leaf enabled {
          type boolean;
          default true;
        }
        leaf speed {
          type uint32;
          units mbps;
          description
            "The advertised speed, in mbps.";
        }
      }
```

```
        leaf speed {
```



```
          type uint32;
          units mbps;
          config false;
          description
            "The speed of the interface, in mbps.";
        }

        list address {
          key ip;

          leaf ip {
            type inet:ip-address;
          }
          leaf prefix-length {
            type uint8;
          }
        }
      }
    }
  }
```

The operator has configured the host name and two interfaces, so the contents of <intended> is:

```
<system xmlns="urn:example:system">

  <hostname>foo</hostname>

  <interface>
    <name>eth0</name>
    <auto-negotiation>
      <speed>1000</speed>
    </auto-negotiation>
    <address>
      <ip>2001:db8::10</ip>
      <prefix-length>32</prefix-length>
    </address>
  </interface>

  <interface>
```

```
      <name>eth1</name>
      <address>
        <ip>2001:db8::20</ip>
        <prefix-length>32</prefix-length>
      </address>
    </interface>

  </system>
```


The system has detected that the hardware for one of the configured interfaces ("eth1") is not yet present, so the configuration for that interface is not applied. Further, the system has received a host name and an additional IP address for "eth0" over DHCP. This is reflected in <applied>:

```
<system
    xmlns="urn:example:system"
    xmlns:arch="urn:ietf:params:xml:ns:yang:ietf-yang-architecture">

  <hostname arch:origin="arch:dynamic">bar</hostname>

  <interface arch:origin="arch:static">
    <name>eth0</name>
    <auto-negotiation>
      <speed>1000</speed>
    </auto-negotiation>
    <address>
      <ip>2001:db8::10</ip>
      <prefix-length>32</prefix-length>
    </address>
    <address arch:origin="arch:dynamic">
      <ip>2001:db8::1:100</ip>
      <prefix-length>32</prefix-length>
    </address>
  </interface>

</system>
```

In <operational-state>, all data from <applied> is present, in addition to a default value, a loopback interface automatically added by the system, and the result of the "speed" auto-negotiation:

```
   <system
       xmlns="urn:example:system"
       xmlns:arch="urn:ietf:params:xml:ns:yang:ietf-yang-architecture">

     <hostname arch:origin="arch:dynamic">bar</hostname>

     <interface arch:origin="arch:static">
       <name>eth0</name>
       <auto-negotiation>
         <enabled arch:origin="arch:data-model">true</enabled>
         <speed>1000</speed>
       </auto-negotiation>
       <speed>100</speed>
       <address>
         <ip>2001:db8::10</ip>
         <prefix-length>32</prefix-length>
       </address>
       <address arch:origin="arch:dynamic">
         <ip>2001:db8::1:100</ip>
         <prefix-length>32</prefix-length>
       </address>
     </interface>

     <interface arch:origin="arch:system">
       <name>lo0</name>
       <address>
```

```
        <ip>::1</ip>
        <prefix-length>128</prefix-length>
      </address>
    </interface>

  </system>
```

Appendix B. Open Issues

1. Do we need another DS <active> inbetween <running> and <intended>? This DS would allow a client to see all active nodes, including unexpanded templates.

2. How do we handle semantical constraints in <operational-state>? Are they just ignored? Do we need a new YANG statement to define if a "must" constraints applies to the <operational-state>?

3. Should it be possible to ask for <applied> in RESTCONF?

4. Better name for "static configuration"?

5. Better name for "intended"?

Authors' Addresses

Martin Bjorklund (editor)
Tail-f Systems

Email: mbj@tail-f.com

Juergen Schoenwaelder
Jacobs University

Email: j.schoenwaelder@jacobs-university.de

Phil Shafer
Juniper

Email: phil@juniper.net

Kent Watsen
Juniper

Email: kwatsen@juniper.net

Rob Wilton
Cisco

Email: rwilton@cisco.com