Workgroup: NETMOD
Internet-Draft:
draft-ma-netmod-immutable-flag-08
Published: 9 July 2023
Intended Status: Standards Track
Expires: 10 January 2024
Authors: Q. Ma (Huawei)
Q. Wu (Huawei)
B. Lengyel (Ericsson)
H. Li (HPE)

YANG Extension and Metadata Annotation for Immutable Flag

Abstract

This document defines a way to formally document existing behavior, implemented by servers in production, on the immutability of some system configuration nodes, using a YANG "extension" and a YANG metadata annotation, both called "immutable", which are collectively used to flag which nodes are immutable.

Clients may use "immutable" statements in the YANG, and annotations provided by the server, to know beforehand when certain otherwise valid configuration requests will cause the server to return an error.

The immutable flag is descriptive, documenting existing behavior, not proscriptive, dictating server behavior.

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 10 January 2024.

Copyright Notice

Table of Contents

1. Introduction

This document defines a way to formally document as a YANG extension or YANG metadata an existing model handling behavior that is already allowed in YANG and has been used by multiple standard organizations and vendors. It is the aim to create one single standard solution for documenting modification restrictions on data declared as configuration, instead of the multiple existing vendor and organization specific solutions. See Appendix B for existing implementations.

YANG [RFC7950] is a data modeling language used to model both state and configuration data, based on the "config" statement. However, there exists some system configuration data that cannot be modified by the client (it is immutable), but still needs to be declared as "config true" to:

- allow configuration of data nodes under immutable lists or containers;
- place "when", "must" and "leafref" constraints between configuration and immutable data nodes.
- ensure the existence of specific list entries that are provided and needed by the system, while additional list entries can be created, modified or deleted;

Client attempts to override an immutable system configuration node are always rejected by the server [I-D.ietf-netmod-system-config]. If the server knows that it will always reject the modification because it internally think it immutable, it should document this towards the clients in a machine-readable way.

This document defines a way to formally document existing behavior, implemented by servers in production, on the immutability of some system configuration nodes, using a YANG "extension" [RFC7950] and a YANG metadata annotation [RFC7952], both called "immutable", which are collectively used to flag which nodes are immutable.

The "immutable" YANG extension is used when the behavior is independent of instances and can be described at the schema-level, while the "immutable" metadata annotation is used when the behavior must be described at the YANG "list" or "leaf-list" instance level.

Comment: Should the "immutable" metadata annotation also be returned for nodes described as immutable in the YANG schema?

Immutability is an existing model handling practice. This document does not apply to the server which does not have any immutable system configuration. While in some cases it may be needed, it also has disadvantages, therefore it SHOULD be avoided wherever possible.

The following is a list of already implemented and potential use cases.

UC1 Modeling of server capabilities

UC2 HW based auto-configuration

UC3 Predefined Access control Rules

UC4 Declaring immutable system configuration from an LNE's perspective

Appendix A describes the use cases in detail.

1.1. Terminology

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.

The following terms are defined in [RFC6241]:

- *configuration data

The following terms are defined in [RFC7950]:

- *data node
- *leaf
- *leaf-list
- *container
- *list
- *anydata
- *anyxml
- *interior node
- *data tree

The following terms are defined in [RFC8341]:

- *access operation

- *write access

The following terms are defined in this document:

immutable flag: A read-only state value the server provides to describe system data it considers immutable. In schema, the immutability of data nodes is conveyed via a YANG "extension" statement. In instance representations, the immutability of data nodes is conveyed via a YANG metadata annotation. Both the extension statement and the metadata annotation are called "immutable". Together, they are alternative ways to express the same behavior.

1.2. Applicability

This document focuses on the configuration which can only be created, updated and deleted by the server, thus cannot be created, updated and deleted by the client.

The "immutable" concept defined in this document only documents existing write access restrictions to writable datastores, given the client is never allowed to edit read-only datastores. The immutable annotation information is also visible even in read-only datastores like <system> (if exists), <intended> and <operational> when a "with-immutable" parameter is carried (see Section 6.2), however this only serves as descriptive information about the instance node itself, but has no effect on the handling of the read-only datastore.

A particular data node or instance has the same immutability in all writable datastores. The immutability of data nodes is protocol and user independent. The immutability and configured value of an existing node must only change by software upgrade or hardware resource/license change.

2. Solution Overview

Immutable configuration can only be created by the system regardless of the implementation of the system configuration datastore [I-D.ietf-netmod-system-config]. If the server implements <system>, immutable configuration is present in <system>. It may be updated or deleted depending on factors like software upgrade or hardware resources/license change. Immutable configuration does not affect the contents of <running> by default.

A client may create/delete immutable nodes with same values as found in <system> (if exists) in read-write configuration datastore (e.g., <running>), which merely mean making immutable nodes visible/ invisible in read-write configuration datastore (e.g., <running>).

If a client tries to override immutable nodes with different values from ones in <system> (if exists), an error is always returned. This document allows the existing immutable system nodes to be formally documented by YANG extension or metadata annotation rather than be written as plain text in the description statement.

Servers reject client's request for updating configuration data when they internally think it immutable. The error reporting is performed immediately at an <edit-config> operation time, regardless what the target configuration datastore is. For an example of an "invalid-value" error response, see Appendix A.2.1.

Servers adding the immutable property which does not have any additional semantic meaning is discouraged. For example, a key leaf that is given a value and cannot be modified once a list entry is created.

The "immutable" flag is intended to be descriptive.

3. Use of "immutable" Flag for Different Statements

This section defines what the immutable flag means to the client for each YANG data node statement. Whilst this section describes immutability at the schema level, it applies equally to when the immutable flag is set via the metadata annotation on node instances.

Throughout this section, the word "change" refers to create, update, and delete.

3.1. The "leaf" Statement

When a leaf node is immutable, its value cannot change.

3.2. The "leaf-list" Statement

When a leaf-list data node is immutable, its value cannot change.

When the "immutable" YANG extension statement is used on a leaf-list data node, or if a leaf-list inherits immutability from an ancestor, it means that the leaf-list as a whole cannot change: entries cannot be added, removed, or reordered, in case the leaf-list is "ordered-by user".

3.3. The "container" Statement

When a container data node is immutable, its instance cannot change, unless the immutability of its descendant node is toggled.

By default, as with all interior nodes, immutability is recursively applied to descendants (see Section 4).

3.4. The "list" Statement

When a list data node is immutable, its instance cannot change, unless the immutability of its descendant node is toggled, per the description elsewhere in this section.

By default, as with all interior nodes, immutability is recursively applied to descendants (see Section 4). This statement is applicable only to the "immutable" YANG extension, as the "list" node does not itself appear in data trees.

3.5. The "anydata" Statement

When an anydata data node is immutable, its instance cannot change. Additionally, as with all interior nodes, immutability is recursively applied to descendants (see Section 4).

Descendants for anydata data node is unknown at module design time, they cannot reset the immutability state with "immutable" YANG extension.

3.6. The "anyxml" Statement

When an "anyxml" data node is immutable, its instance cannot change. Additionally, as with all interior nodes, immutability is recursively applied to descendants (see Section 4).

Descendants for anyxml data node is unknown at module design time, they cannot reset the immutability state with "immutable" YANG extension.

4. Immutability of Interior Nodes

Immutability is a conceptual operational state value that is recursively applied to descendants, which may reset the immutability state as needed, thereby affecting their descendants. There is no limit to the number of times the immutability state may change in a data tree.

For example, given the following application configuration XML snippets:

```
<application im:immutable="true">
  <name>predefined-ftp</name>
  <protocol>ftp</protocol>
  <port-number im:immutable="false">69</port-number>
</application>
```

The list entry named "predefined-ftp" is immutable="true", but its child node "port-number" has the immutable="false" (thus the client can override this value). The other child node (e.g., "protocol") not specifying its immutability explicitly inherits immutability from its parent node thus is also immutable="true".

5. "Immutable" YANG Extension

5.1. Definition

If servers always reject client modification attempts to some data node that they internally think immutable and irrelevant to its instance data, an "immutable" YANG extension can be used to formally indicate to the clients.

The "immutable" YANG extension can be a substatement to a "config true" leaf, leaf-list, container, list, anydata or anyxml statement. It has no effect if used as a substatement to a "config false" node, but can be allowed anyway.

The "immutable" YANG extension defines an argument statement named "value" which is a boolean type to indicate that whether the node is immutable or not. If the "immutable" YANG extension is not specified for a particular data node, the default immutability is the same as that of its parent node. The immutability for a top-level data node is "false" by default.

6. "Immutable" Metadata Annotation

6.1. Definition

If servers always reject clients modification to some particular instance that they internally think immutable, an "immutable" metadata annotation can be used to formally indicate to the clients.

The "immutable" metadata annotation takes as an value which is a boolean type, it is not returned unless a client explicitly requests through a "with-immutable" parameter (see Section 6.2). If the "immutable" metadata annotation for data node instances is not specified, the default "immutable" value is the same as the immutability of its parent node in the data tree. The immutable metadata annotation value for a top-level instance node is false if not specified.

Note that "immutable" metadata annotation is used to annotate data node instances. A list may have multiple entries/instances in the data tree, "immutable" can annotate some of the instances as read-only, while others are read-write.

6.2. "with-immutable" Parameter

The YANG model defined in this document (see Section 8) augments the <get-config>, <get> operation defined in RFC 6241, and the <get-data> operation defined in RFC 8526 with a new parameter named "with-immutable". When this parameter is present, it requests that the server includes "immutable" metadata annotations in its response.

This parameter may be used for read-only configuration datastores, e.g., <system> (if exists), <intended> and <operational>, but the "immutable" metadata annotation returned indicates the immutability towards read-write configuration datastores, e.g., <startup>, <candidate> and <running>. If the "immutable" metadata annotation for returned child nodes are omitted, it has the same immutability as its parent node. The immutability of top hierarchy of returned nodes is false by default.

Note that "immutable" metadata annotation is not included in a response unless a client explicitly requests them with a "with-immutable" parameter.

7. Interaction between Immutable Flag and NACM

The server rejects an operation request due to immutability when it tries to perform the operation on the request data. It happens after any access control processing, if the Network Configuration Access Control Model (NACM) [RFC8341]is implemented on a server. For example, if an operation requests to override an immutable configuration data, but the server checks the user is not authorized to perform the requested access operation on the request data, the request is rejected with an "access-denied" error.

8. YANG Module

```
<CODE BEGINS> file="ietf-immutable@2023-07-09.yang"
//RFC Ed.: replace XXXX with RFC number and remove this note
  module ietf-immutable {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-immutable";
    prefix im;

    import ietf-yang-metadata {
      prefix md;
    }
    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 Network Modeling (NETMOD) 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: Balazs Lengyel
               <mailto:balazs.lengyel@ericsson.com>

       Author: Hongwei Li
               <mailto:flycoolman@gmail.com>";

    description
      "This module defines a YANG extension and a metadata annotation
       both called 'immutable', to allow the server to formally
       document existing behavior on the mutability of some
       configuration nodes. Clients may use 'immutable' extension
       statements in the YANG, and annotations provided by the server
       to know beforehand when certain otherwise valid configuration
       requests will cause the server to return an error.
```

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

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

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

   The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
   'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
   'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
   are to be interpreted as described in BCP 14 (RFC 2119)
   (RFC 8174) when, and only when, they appear in all
   capitals, as shown here.";

 revision 2023-05-25 {
   description
     "Initial revision.";
   // RFC Ed.: replace XXXX and remove this comment
   reference
     "RFC XXXX: YANG Extension and Metadata Annotation for
      Immutable Flag";
 }

 extension immutable {
   argument value;
   description
     "If servers always reject client modification attempts to
      some data node that can only be created, modified and
      deleted by the device itself, an 'immutable' YANG extension
      can be used to formally indicate to the client.

      The statement MUST only be a substatement to a 'config true'
      leaf, leaf-list, container, list, anydata or anyxml
      statement. Zero or one immutable statement per parent
      statement is allowed.

      No substatements are allowed.

      The argument of the 'immutable' statement defines the value,
      indicating whether the node is immutable or not.

      Adding immutable of an existing immutable statement
      is non-backwards compatible changes.
```

```
     Other changes to immutable are backwards compatible.";
}

md:annotation immutable {
  type boolean;
  description
    "If servers always reject clients modification to some
     particular instance that can only be created, modified and
     deleted by the device itself, an 'immutable' metadata
     annotation can be used to formally indicate to the clients.
     The 'immutable' annotation indicates the immutability of an
     instantiated data node.

     The 'immutable' metadata annotation takes as a value 'true'
     or 'false'. If the 'immutable' metadata annotation for data
     node instances is not specified, the default value is false.
     Explicitly annotating instances as immutable=true has the
     same effect as not specifying this value.";
}

grouping with-immutable-grouping {
  description
    "define the with-immutable grouping.";
  leaf with-immutable {
    type empty;
    description
      "If this parameter is present, the server will return the
       'immutable' annotation for configuration that it
       internally thinks it immutable. When present, this
       parameter allows the server to formally document existing
       behavior on the mutability of some configuration nodes.";
  }
}
augment "/ncds:get-data/ncds:input" {
  description
    "Allows the server to include 'immutable' metadata
     annotations in its response to get-data operation.";
  uses with-immutable-grouping;
}
augment "/nc:get-config/nc:input" {
  description
    "Allows the server to include 'immutable' metadata
     annotations in its response to get-config operation.";
  uses with-immutable-grouping;
}
augment "/nc:get/nc:input" {
  description
    "Allows the server to include 'immutable' metadata
     annotations in its response to get operation.";
```

```
      uses with-immutable-grouping;
    }
  }
```

<CODE ENDS>

9. IANA Considerations

9.1. The "IETF XML" Registry

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

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

9.2. The "YANG Module Names" Registry

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

name: ietf-immutable
prefix: im
namespace: urn:ietf:params:xml:ns:yang:ietf-immutable
RFC: XXXX
// RFC Ed.: replace XXXX and remove this comment

10. Security Considerations

The YANG module specified in this document defines a YANG extension and a metadata Annotation. These can be used to further restrict write access but cannot be used to extend access rights.

This document does not define any protocol-accessible data nodes.

Since immutable information is tied to applied configuration values, it is only accessible to clients that have the permissions to read the applied configuration values.

The security considerations for the Defining and Using Metadata with YANG (see Section 9 of [RFC7952]) apply to the metadata annotation defined in this document.

Acknowledgements

Thanks to Kent Watsen, Andy Bierman, Robert Wilton, Jan Lindblad, Reshad Rahman, Anthony Somerset, Lou Berger, Joe Clarke, Scott Mansfield for reviewing, and providing important input to, this document.

References

Normative References

[RFC2119]

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

[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>.

[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>.

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

[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>.

Informative References

[I-D.ietf-netmod-system-config] Ma, Q., Wu, Q., and C. Feng, "System-defined Configuration", Work in Progress, Internet-Draft, draft-ietf-netmod-system-config-02, 4 July 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-system-config-02>.

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

[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>.

[RFC8530] Berger, L., Hopps, C., Lindem, A., Bogdanovic, D., and X. Liu, "YANG Model for Logical Network Elements", RFC 8530,

DOI 10.17487/RFC8530, March 2019, <https://www.rfc-editor.org/info/rfc8530>.

[TR-531] ONF, "UML to YANG Mapping Guidelines, <https://wiki.opennetworking.org/download/attachments/376340494/Draft_TR-531_UML-YANG_Mapping_Gdls_v1.1.03.docx?version=5&modificationDate=1675432243513&api=v2>", February 2023.

[TS28.623] 3GPP, "Telecommunication management; Generic Network Resource Model (NRM) Integration Reference Point (IRP); Solution Set (SS) definitions, <https://www.3gpp.org/ftp/Specs/archive/28_series/28.623/28623-i02.zip>".

[TS32.156] 3GPP, "Telecommunication management; Fixed Mobile Convergence (FMC) Model repertoire, <https://www.3gpp.org/ftp/Specs/archive/32_series/32.156/32156-h10.zip>".

Appendix A. Detailed Use Cases

A.1. UC1 - Modeling of server capabilities

System capabilities might be represented as system-defined data nodes in the model. Configurable data nodes might need constraints specified as "when", "must" or "path" statements to ensure that configuration is set according to the system's capabilities. E.g.,

- A timer can support the values 1,5,8 seconds. This is defined in the leaf-list 'supported-timer-values'.

- When the configurable 'interface-timer' leaf is set, it should be ensured that one of the supported values is used. The natural solution would be to make the 'interface-timer' a leaf-ref pointing at the 'supported-timer-values'.

However, this is not possible as 'supported-timer-values' must be read-only thus config=false while 'interface-timer' must be writable thus config=true. According to the rules of YANG it is not allowed to put a constraint between config true and false data nodes.

The solution is that the supported-timer-values data node in the YANG Model shall be defined as "config true" and shall also be marked with the "immutable" extension making it unchangable. After this the 'interface-timer' shall be defined as a leaf-ref pointing at the 'supported-timer-values'.

A.2. UC2 - HW based auto-configuration - Interface Example

This section shows how to use immutable YANG extension to mark some data node as immutable.

When an interface is physically present, the system will create an interface entry automatically with valid name and type values in <system> (if exists, see [I-D.ietf-netmod-system-config]). The system-generated data is dependent on and must represent the HW present, and as a consequence must not be changed by the client. The data is modelled as "config true" and should be marked as immutable.

Seemingly an alternative would be to model the list and these leaves as "config false", but that does not work because:

* The list cannot be marked as "config false", because it needs to contain configurable child nodes, e.g., ip-address or enabled;

* The key leaf (name) cannot be marked as "config false" as the list itself is config true;

* The type cannot be marked "config false", because we MAY need to reference the type to make different configuration nodes conditionally available.

The immutability of the data is the same for all interface instances, thus following fragment of a fictional interface module including an "immutable" YANG extension can be used:

```
container interfaces {
  list interface {
    key "name";
    leaf name {
      type string;
    }
    leaf type {
      im:immutable;
      type identityref {
        base ianaift:iana-interface-type;
      }
      mandatory true;
    }
    leaf mtu {
      type uint16;
    }
    leaf-list ip-address {
      type inet:ip-address;
    }
  }
}
```

Note that the "name" leaf is defined as a list key which can never been modified for a particular list entry, there is no need to mark "name" as immutable.

A.2.1. Error Response to Client Updating the Value of an Interface Type

This section shows an example of an error response due to the client modifying an immutable configuration.

Assume the system creates an interface entry named "eth0" given that an inerface is inserted into the device. If a client tries to change the type of an interface to a value that doesn't match the real type of the interface used by the system, the request will be rejected by the server:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <interface xc:operation="merge"
            xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
        <name>eth0</name>
        <type>ianaift:tunnel</type>
      </interface>
    </config>
  </edit-config>
</rpc>

<rpc-reply message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>invalid-value</error-tag>
    <error-severity>error</error-severity>
    <error-path xmlns:t="http://example.com/schema/1.2/config">
      /interfaces/interface[name="eth0"]/type
    </error-path>
    <error-message xml:lang="en">
      Invalid type for interface eth0
    </error-message>
  </rpc-error>
</rpc-reply>
```

A.3. UC3 - Predefined Access control Rules

Setting up detailed rules for access control is a complex task. (see [RFC8341]) A vendor may provide an initial, predefined set of groups and related access control rules so that the customer can use access control out-of-the-box. The customer may continue using these predefined rules or may add his own groups and rules. The predefined groups shall not be removed or altered guaranteeing that access control remains usable and basic functions e.g., a system-security-administrator are always available.

The system needs to protect the predefined groups and rules, however, the list "groups" or the list "rule-list" cannot be marked as config=false or with the "immutable" extension in the YANG model because that would prevent the customer adding new entries. Still it would be good to notify the client in a machine readable way that the predefined entries cannot be modified. When the client retrieves access control data the immutable="true" metadata annotation should be used to indicate to the client that the predefined groups and rules cannot be modified.

A.4. UC4 - Declaring immutable system configuration from an LNE's perspective

An LNE (logical network element) is an independently managed virtual network device made up of resources allocated to it from its host or parent network device [RFC8530]. The host device may allocate some resources to an LNE, which from an LNE's perspective is provided by the system and may not be modifiable.

For example, a host may allocate an interface to an LNE with a valid MTU value as its management interface, so that the allocated interface should then be accessible as the LNE-specific instance of the interface model. The assigned MTU value is system-created and immutable from the context of the LNE.

Appendix B. Existing implementations

There are already a number of full or partial implementations of immutability.

3GPP TS 32.156 [TS32.156] and 28.623 [TS28.623]: Requirements and a partial solution

ITU-T using ONF TR-531[TR-531] concept on information model level but no YANG representation.

Ericsson: requirements and solution

YumaPro: requirements and solution

Nokia: partial requirements and solution

Huawei: partial requirements and solution

Cisco using the concept at least in some YANG modules

Junos OS provides a hidden and immutable configuration group called junos-defaults

Appendix C. Changes between revisions

Note to RFC Editor (To be removed by RFC Editor)

v06 - v07

* Use a Boolean type for the immutable value in YANG extension and metadata annotation

* Define a "with-immutable" parameter and state that immutable metadata annotation is not included in a response unless a client explicitly requests them with a "with-immutable" parameter

* reword the abstract and related introduction section to highlight immutable flag is descriptive

* Add a new section to define immutability of interior nodes, and merge with "Inheritance of Immutable configuration" section

* Add a new section to define what the immutable flag means for each YANG data node

* Define the "immutable flag" term.

* Add an item in the open issues tracking: Should the "immutable" metadata annotation also be returned for nodes described as immutable in the YANG schema so that there is a single source of truth?

v05 - v06

* Remove immutable BGP AS number case

* Fix nits

v04 - v05

* Emphasized that the proposal tries to formally document existing allowed behavior

* Reword the abstract and introduction sections;

*Restructure the document;

*Simplified the interface example in Appendix;

*Add immutable BGP AS number and peer-type configuration example.

*Added temporary section in Appendix B about list of existing non-standard solutions

*Clarified inheritance of immutability

*Clarified that this draft is not dependent on the existence of the <system> datastore.

v03 - v04

*Clarify how immutable flag interacts with NACM mechanism.

v02 - v03

*rephrase and avoid using "server MUST reject" statement, and try to clarify that this documents aims to provide visibility into existing immutable behavior;

*Add a new section to discuss the inheritance of immutability;

*Clarify that deletion to an immutable node in <running> which is instantiated in <system> and copied into <running> should always be allowed;

*Clarify that write access restriction due to general YANG rules has no need to be marked as immutable.

*Add an new section named "Acknowledgements";

*editoral changes.

v01 - v02

*clarify the relation between the creation/deletion of the immutable data node with its parent data node;

*Add a "TODO" comment about the inheritance of the immutable property;

*Define that the server should reject write attempt to the immutable data node at an <edit-config> operation time, rather than waiting until a <commit> or <validate> operation takes place;

v00 - v01

*Added immutable extension

*Added new use-cases for immutable extension and annotation

*Added requirement that an update that means no effective change should always be allowed

*Added clarification that immutable is only applied to read-write datastore

*Narrowed the applied scope of metadata annotation to list/leaf-list instances

Appendix D. Open Issues tracking

*Should the "immutable" metadata annotation also be returned for nodes described as immutable in the YANG schema so that there is a single source of truth?

Authors' Addresses

Qiufang Ma
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

Balazs Lengyel
Ericsson

Email: balazs.lengyel@ericsson.com

Hongwei Li
HPE

Email: flycoolman@gmail.com