Workgroup: NETMOD
Internet-Draft:
draft-ma-netmod-immutable-flag-01
Published: 14 April 2022
Intended Status: Standards Track
Expires: 16 October 2022
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 YANG extension named "immutable" to indicate that specific data nodes are not allowed to be created/deleted/updated. To indicate that specific instances of a list/leaf-list node cannot be changed after initialization, a metadata annotation with the same name is also defined. Any data node or instance marked as immutable is read-only to the clients of YANG-driven management protocols, such as NETCONF, RESTCONF and other management operations (e.g., SNMP and CLI requests).

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 16 October 2022.

Copyright Notice

Table of Contents

1. Introduction

YANG [RFC7950] is a data modeling language used to model both state and configuration data, based on the "config" statement. However there exists data that should not be modifiable by the client, but still needs to be declared as "config true" to:

- allow configuration of data nodes under immutable lists or containers;
- 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;
- place "when", "must" and "leafref" constraints between configuration and immutable schema nodes.

E.g., the interface name and type values created by the system due to the hardware currently present in the device cannot be modified by clients, while configurations such as MTU created by the system are free to be modified by the client. Further examples and use-cases are described in Appendix A.

Allowing some configuration to be modifiable while other parts are not is inconsistent and introduces ambiguity to clients.

To address this issue, this document defines a YANG extension and a metadata annotation [RFC7952] named "immutable" to indicate the immutability characteristic of a particular schema node or instantiated data node. If a schema node is marked as immutable, data nodes based on the schema MUST NOT be added, removed or updated by management protocols, such as NETCONF, RESTCONF or other management operations (e.g., SNMP and CLI requests). If an instantiated data node is marked as immutable the server MUST reject changes to it by YANG-driven management protocols, such as NETCONF, RESTCONF and other management operations (e.g., SNMP and CLI requests). Marking instance data nodes as immutable (as opposed to marking schema-nodes) is important when only some instances of a list or leaf-list shall be marked as read-only.

Theoretically, any "config true" data node is allowed to be created, updated and deleted. This work makes write access restrictions other than general YANG and NACM rules visible, which doesn't mean attaching such restrictions is encouraged.

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] and [RFC8341] and are not redefined here:

- *configuration data
- *access operation
- *write access

The following terms are defined in this document:

immutable: A property indicating that a schema node is not allowed to be created/deleted/updated. When annotating an instance of a list/leaf-list, it indicates that the instance cannot be updated once it's created. The immutability of a specific data node or instance is datastore-independent, protocol-independent and user-independent.

2. Overview

The "immutable" concept only puts write access restrictions to read-write datastores. When a specific data node or instance is marked as "immutable", NACM cannot override this to allow create/delete/update access.

Immutability is a property of the object itself. A particular data node or instance MUST have the same immutability in all read-write datastores. The immutable property should be visible even in read-only datastores (e.g., <system>, <intended>, <operational>), however this only serves as information about the data node itself, but has no effect on the handling of the read-only datastore. In addition, the immutability property of a particular data node or instance MUST NOT change due to different network management protocols and users.

If a particular list/leaf-list node is marked as "immutable" without exceptions for "update" in the schema (e.g., a list data node is always immutable and an update is not allowed), the server SHOULD NOT annotate its instances duplicately.

Servers MUST reject any attempt to the "create", "delete" and "update" access operations on an immutable data node or instance; marked by YANG extension (except according to the exceptions argument) or metadata annotation. The error reporting is performed at various different time according to the selected read-write target datastore. If the target datastore is "running", the server should reply with an "invalid-value" at a <edit-config> operation time. If the target datastore is "candidate", the "invalid-value" error response to update an immutable data node is delayed until a <commit> or <validate> operation takes place. For an example of an "invalid-value" error response, see Appendix A.1.2.

However the following operations SHOULD be allowed:

* Use a create, update, delete/remove operation on an immutable node/instance if the effective change is null. E.g. If a leaf has a current value of "5" it should be allowed to replace it with a value of "5".

* Create an immutable data node/instance with a same value initially set by the system if it doesn't exist in the datastore, e.g., explicitly configure a system generated interface name and type in <running>;

3. "Immutable" YANG Extension

The "immutable" YANG extension can be a substatement to a leaf, leaf-list, container, list, anydata or anyxml statement. It indicates that data nodes based on the parent statement MUST NOT be

added, removed or updated except according to the exceptions argument. The server MUST reject any such write attempt.

The "immutable" YANG extension defines an argument statement named "exceptions" which gives a list of operations that users are permitted to invoke for the specified node.

The following values are supported for the "exceptions" argument:

- *Create: allow users to create instances of the data node;

- *Update: allow users to modify instances of the data node;

- *Delete: allow users to delete instances of the data node.

4. "Immutable" Metadata Annotation

The "immutable" flag is used to indicate the immutability of a particular instantiated data node. It only applies to the list/leaf-list entries. The values are boolean types indicating whether the data node instance is immutable or not.

Any list/leaf-list instance annotated with immutable="true" is read-only to clients, which means that once an instance is created, the client cannot change it. If a list entry is annotated with immutable="true", any contained descendant instances of any type (including leafs, lists, containers, etc.) inside the specific instance is not allowed to be created, updated and deleted without the need to annotate descendant nodes instances explicitly.

Note that "immutable" metadata annotation is used to annotate instances of a list/leaf-list rather than schema nodes. For instance, a list node may exist in multiple instances in the data tree, "immutable" can annotate some of the instances as read-only, while others are not.

When the client retrieves a particular datastore, immutable data node instances MUST be annotated with immutable="true" by the server. If the "immutable" metadata annotation inside a list entry is not specified, the default "immutable" value for a list/leaf-list entry is false.

Different from the "immutable" YANG extension, deletion to an instance marked with immutable="true" metadata annotation SHOULD always be allowed unless the list/leaf-list data node in the schema has an im:immutable extension as substatement without a "delete" exception.

5. YANG Module

```
<CODE BEGINS> file="ietf-immutable@2022-03-28.yang"
  module ietf-immutable {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-immutable";
    prefix im;

    import ietf-yang-metadata {
      prefix md;
    }

    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 metadata annotation named 'immutable'
       to indicate the immutability of a particular instantiated
       data node. Any instantiated data node marked with
       immutable='true' by the server is read-only to the clients
       of YANG-driven management protocols, such as NETCONF,
       RESTCONF as well as SNMP and CLI requests.

       The module defines the immutable extension that indicates
       that data nodes based ona data-dafinition statement cannot
       be added removed or updated except according to the
       exceptions argument.

       Copyright (c) 2022 IETF Trust and the persons identified
       as authors of the code. All rights reserved.

       Redistribution and use in source and binary forms, with
       or without modification, is permitted pursuant to, and
       subject to the license terms contained in, the Simplified
       BSD License set forth in Section 4.c of the IETF Trust's
```

```
   Legal Provisions Relating to IETF Documents
   (https://trustee.ietf.org/license-info).

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

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

revision 2022-03-28 {
  description
    "Initial revision.";
  reference
    "RFC XXX: Immutable Metadata Annotation";
}

extension immutable {
  argument exceptions;
  description
    "The 'immutable' extension as a substatement to a data
     definition statement indicates that data nodes based on
     the parent statement MUST NOT be added, removed or
     updated by management protocols, such as NETCONF,
     RESTCONF or other management operations (e.g., SNMP
     and CLI requests) except when indicated by the
     exceptions argument.

     Immutable data MAY be marked as config true to allow
     'leafref', 'when' or 'must' constraints to be based
     on it.

     The statement MUST only be a substatement of the leaf,
     leaf-list, container, list, anydata, anyxml statements.
     Zero or one immutable statement per parent statement
     is allowed.
     NO substatements are allowed.

     The argument is a list of operations that users are
     permitted to be used for the specified node, while
     other operations are forbidden by the immutable extension.
     - create: allows users to create instances of the object
     - update : allows users to modify instances of the object
     - delete : allows users to delete instances of the object

     To dis-allow all user write access, omit the argument;
```

```
        To allow only create and delete user access, provide
        the string 'create delete' for the 'exceptions' parameter.

        Providing all 3 parameters has the same affect as not
        using this extension at all, but can be used anyway.

        Equivalent YANG definition for this extension:

        leaf immutable {
          type bits {
            bit create;
            bit update;
            bit delete;
          }
          default '';
        }

        Adding immutable or removing values from the
        exceptions argument of an existing immutable statement
        are non-backwards compatible changes.
        Other changes to immutable are backwards compatible.";
    }

    md:annotation immutable {
      type boolean;
      description
        "The 'immutable' annotation indicates the immutability of an
         instantiated data node. Any data node instance marked as
         'immutable=true' is read-only to clients and cannot be
         updated through NETCONF, RESTCONF or CLI. It applies to the
         list and leaf-list entries. The default is 'immutable=false'
         if not specified for an instance.";
    }
  }

<CODE ENDS>
```

6. IANA Considerations

6.1. 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.

6.2. 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
```

7. Security Considerations

The YANG module specified in this document defines a metadata annotation for data nodes that is designed to be accessed network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

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.

8. References

8.1. 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>.

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

[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>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[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>.

[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>.

[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>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

8.2. Informative References

[I-D.ma-netmod-with-system] Ma, Q., Watsen, K., Wu, Q., Chong, F., and J. Lindblad, "System-defined Configuration", Work in Progress, Internet-Draft, draft-ma-netmod-with-system-03, 10 April 2022, <https://www.ietf.org/archive/id/draft-ma-netmod-with-system-03.txt>.

[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>.

Appendix A. Usage Examples

A.1. Interface Example

This section shows how to use im: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> (see [I-D.ma-netmod-with-system]). 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 immuable.

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 "create";
      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.1.1. Creating an Interface with a "type" Value

As defined in the YANG model, there is an exception for "create" operation. Assume the interface hardware is not present physically

at this point, the client is allowed to create an interface named "eth0" with a type value in <running>:

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

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

The interface data does not appear in <operational> since the physical interface doesn't exist. When the interface is inserted, the system will detect it and create the associated configuration in <system>. The system tries to merge the interface configuration in the <running> datastore with the same name as the inserted interface configuration in <system>. If no such interface configuration named "eth0" is found in <system> or the type set by the client doesn't match the real interface type generated by the system, only the system-defined interface configuration is applied and present in <operational>.

A.1.2. Updating the Value of an Interface Type

Assume the system applied the interface configuration named "eth0" successfully. 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 server must reject the request:

```
<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.2. Immutable System Capabilities Modelled as "config true"

System capabilities might be represented as system-set 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 readOnly 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 schema 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. After this the 'interface-timer' shall be defined as a leaf-ref pointing at the 'supported-timer-values'.

A.3. Immutable System-defined List Entries

There are some system-defined entries for a "config true" list which are present in <system> (see [I-D.ma-netmod-with-system]) and cannot be updated by the client, such system-defined instances should be defined immutable. The client is free to define, update and delete their own list entries in <running>. Thus the list data node in the YANG model cannot be marked as "immutable" extension as a whole. But some of the system-defined list entries need to be protected if they are copied from the <system> datastore to <running>.

An immutable metadata annotation can be useful in this case. When the client retrieves those system-defined entries towards <system> (or <running> if they are copied into <running>), an immutable="true" annotation is returned; so that the client can understand that the predefined list entries shall not be updated but they can configure their list entries without any restriction.

Appendix B. Changes between revisions

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

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

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