netmod
Internet-Draft
Updates: 8040, 8526 (if approved)
Intended status: Standards Track
Expires: 16 July 2026

Q. Ma, Ed.
Q. Wu
Huawei
B. Lengyel, Ed.
Ericsson
H. Li
HPE

12 January 2026

YANG Metadata Annotation for Immutable Flag
draft-ietf-netmod-immutable-flag-07

Abstract

This document defines a way to formally document an existing behavior, implemented by servers in production, on the immutability of some system-provided nodes, using a YANG metadata annotation called "immutable" to flag which nodes are immutable.

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

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

This document updates RFC 8040 and RFC 8526.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Network Modeling Working Group mailing list (netmod@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/netmod/.

Source for this draft and an issue tracker can be found at https://github.com/netmod-wg/immutable-flag.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Copyright Notice

Table of Contents

1. Introduction

This document defines a YANG metadata annotation [RFC7952] to formally document an existing model handling behavior that has been used by multiple standard organizations and vendors. It is the aim to create one single standard solution for documenting non-modifiable system data declared as configuration, instead of the multiple existing vendor and organization specific solutions.

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 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.

If the server always rejects a client’s attempt to override some system-provided data because it internally thinks immutable, it should document it towards the clients in a machine-readable way rather than writing as plain text in the "description" statement.

This document defines a way to formally document the existing behavior, implemented by servers in production, on the immutability of some system-provided nodes, using a YANG metadata annotation [RFC7952] called "immutable" to flag which nodes are immutable. This document does not regulate server behaviors. That said, it is expected that a server will return an error with an error-tag containing "invalid-value" when immutability is attempted to be violated.

This document does not apply to the server not having any immutable system configuration. While in some cases immutability 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 Hardware based auto-configuration

* UC3 Predefined administrator roles

* UC4 Declaring immutable system configuration from the perspective of a logical network element (LNE)

Appendix A describes the use cases in detail.

1.1. Updates to RFC 8040

This document updates Sections 4.8 and 9.1.1 of [RFC8040] to add an additional input parameter named "with-immutability", as specified in Section 4.2.2.

1.2. Updates to RFC 8526

This document updates Section 3.1.1 of [RFC8526] to add an additional input parameter named "with-immutability" for the <get-data> operation, as specified in Section 4.2.1.

1.3. Editorial Note (To be removed by RFC Editor)

Note to the RFC Editor: This section is to be removed prior to publication.

This document contains placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. No other RFC Editor instructions are specified elsewhere in this document.

Please apply the following replacements:

* XXXX --> the assigned RFC number for this draft

* 2026-01-12 --> the actual date of the publication of this document

2. Conventions and Definitions

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 document uses the following definition in [RFC6241]:

* configuration data

The document uses the following definition in [RFC7950]:

* data node

* leaf

* leaf-list

* container

* list

* anydata

* anyxml

* interior node

* data tree

The document uses the following definition in [RFC8341]:

* access operation

The document uses the following definition in [I-D.ietf-netmod-system-config]:

* system configuration

This document defines the following term:

immutable flag: A read-only state value the server provides to describe immutability of the configuration, which is conveyed via a YANG metadata annotation called "immutable" with a boolean value.

3. Applicability

While immutable flag applies to all configuration nodes, its value "true" can only be used for system configuration.

The immutable flag is only visible in read-only datastores (i.e., <system> [I-D.ietf-netmod-system-config], <intended>, and <operational>) when a "with-immutability" parameter is carried (Section 4.2), however this only serves as descriptive information about the instance node itself, but has no effect on the handling of the read-only datastore. If the immutable flag is requested to be returned for an invalid datastore, then the server MUST return an <rpc-error> element with an <error-tag> value of "invalid-value".

An instance has the same immutability if it appears in different datastores, the immutability of configuration data is also protocol and user independent. The immutability of any configuration data, and the value of any immutable configured data node, MUST only change via software upgrade, hardware resources change, or license change.

4. "Immutable" Metadata Annotation

4.1. Definition

The immutable flag which is defined as the metadata annotation takes a boolean value, and it is returned as requested by the client using a "with-immutability" parameter (Section 4.2). If the "immutable" metadata annotation for a configuration node is not specified, the default "immutable" value is the same as the value of its parent node in the data tree (Section 6). The immutable metadata annotation value for a top-level instance node is "false" if not specified.

A node that is annotated as immutable cannot be changed via configuring a different value in read-write configuration datastores (e.g., <running>), nor is there any way to delete the node from the combined configuration in the intended datastore (as described in Section 4 of [I-D.ietf-netmod-system-config]). The node MAY be explicitly configured by a client in <running> with the same value and that configuration in <running> may subsequently be removed, but neither of these edits will change the configuration in <intended> (if implemented) on the device.

Note that "immutable" metadata annotations are used to annotate data node instances. A list may have multiple instances in the data tree, servers may annotate some of the instances as immutable, while others as mutable.

Servers MUST ignore any immutable annotations sent from the client.

4.2. "with-immutability" Parameter

This section specifies the NETCONF [RFC6241] [RFC8526] and RESTCONF [RFC8040] protocol extensions to support the "with-immutability" parameter. The "immutable" metadata annotations are not returned in a response unless explicitly requested by the client using this parameter.

4.2.1. NETCONF Extensions to Support "with-immutability"

This document updates [RFC8526] to augment the <get-data> operation with an additional parameter named "with-immutability" when interacting with read-only datastores. If present, this parameter requests that the server includes the "immutable" metadata annotations in its response.

Figure 1 provides the tree structure [RFC8340] of augmentations to NETCONF operations, as defined in the "ietf-immutable-annotation" module (Section 9).

```
module: ietf-immutable-annotation
  augment /ncds:get-data/ncds:input:
    +---w with-immutability?   empty
```

Figure 1: Augmentations to NETCONF Operations

Refer to Appendix B.1 for an example of NETCONF operation with "with-immutability" input parameter.

4.2.2. RESTCONF Extensions to Support "with-immutability"

This document extends Sections 4.8 and 9.1.1 of [RFC8040] to add a query parameter named "with-immutability" to the GET operation. If present, this parameter requests that the server includes the "immutable" metadata annotations in its response. This parameter is only allowed with no values carried when interacting with read-only datastores. If it has any unexpected value, then a "400 Bad Request" status-line is returned. RESTCONF protocol operations for the datastore resources are defined in [RFC8527].

To enable a RESTCONF client to discover if the "with-immutability" query parameter is supported by the server, the following capability URI is defined:

urn:ietf:params:restconf:capability:with-immutability:1.0

Refer to Appendix B.2 for an example of RESTCONF operation with "with-immutability" query parameter.

5. Use of Immutable Flag for Different Statements

This section defines what the immutable flag means to the client for each instance of YANG data node statement.

5.1. The "leaf" Statement

When a leaf node instance is immutable, it cannot be configured with a different value in read-write configuration datastores (e.g., <running>) or removed from <intended> (if implemented). Though it can be created/deleted in read-write configuration datastores (see Sections 4.1 and 7).

5.2. The "leaf-list" Statement

When a leaf-list entry is immutable, it cannot be configured with a different value in read-write configuration datastore (e.g., <running>) or removed from <intended> (if implemented). Though it can be created/deleted in read-write configuration datastores (see Sections 4.1 and 7).

The immutable annotation attached to the individual leaf-list entry provides immutability with respect to the entry itself. As per the restrictions in [RFC7952], annotations cannot be attached to an entire leaf-list instance and only to individual leaf-list entries, which implies a leaf-list as a whole can only inherit immutability from a parent node (e.g., container).

If a leaf-list as a whole is immutable, any leaf-list entries cannot be added, modified, or reordered (if it is ordered-by user).

Refer to Appendix B.5 for an example of immutability of leaf-lists.

5.3. The "container" Statement

When a container node instance is immutable, it cannot be removed from <intended> (if implemented). Though it can be created/deleted in read-write configuration datastores (see Sections 4.1 and 7).

Descendant nodes of the container recursively inherit the immutability of the container, unless the immutability is overridden by an "immutable" annotation on a descendant node.

By default, as with all interior nodes, immutability is recursively applied to descendants (Section 6).

5.4. The "list" Statement

When a list entry is immutable, it cannot be removed from <intended> (if implemented). Though it can be created/deleted in read-write configuration datastores (see Sections 4.1 and 7).

Descendant nodes of the list entry recursively inherit the immutability of the list entry, unless the immutability is overridden by an "immutable" annotation on a descendant node.

By default, as with all interior nodes, immutability is recursively applied to descendants (Section 6).

The immutable annotation attached to the individual list entry provides immutability with respect to the entry itself. As per the restrictions in [RFC7952], annotations cannot be attached to an entire list instance and only to individual list entries, which implies a list as a whole can only inherit immutability from a parent node (e.g., container).

If a list as a whole is immutable, any list entries cannot be added, removed, or reordered (if it is ordered-by user). Each list entry inherits the immutability of the list by default, unless the immutability is overridden by an "immutable" annotation on a list entry.

Refer to Appendix B.4 for an example of immutability of lists.

5.5. The "anydata" Statement

When an "anydata" node instance is immutable, it cannot be removed from <intended> (if implemented). Though it can be created/deleted in read-write configuration datastores (see Sections 4.1 and 7).

Additionally, as with all interior nodes, immutability is recursively applied to descendants (Section 6).

5.6. The "anyxml" Statement

When an "anyxml" node instance is immutable, it cannot be removed from <intended> (if implemented). Though it can be created/deleted in read-write configuration datastores (see Sections 4.1 and 7).

Additionally, as with all interior nodes, immutability is recursively applied to descendants (Section 6).

6. 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.

If the "immutable" metadata annotation for returned child node is omitted, it has the same immutability as its parent node. The immutability of top hierarchy of returned nodes is false by default. Servers may suppress the annotation if it is inherited from its parent node or uses the default value as the top-level node, but are not precluded from returning the annotation on every single element.

Refer to Appendix B.3 for an example of how immutability is recursively inherited or explicitly reset by descendants.

7. System Configuration Datastore Interactions

Immutable configuration can only be created, updated and deleted by the server, and it is present in <system>, if implemented. That said, the existence of immutable configuration is independent of whether <system> is implemented or not. Not all system configuration data is immutable. Immutable configuration does not appear in <running> unless it is explicitly configured.

As specified in Section 4.1, a client MAY create/delete immutable nodes with same values as defined by server in read-write configuration datastore (e.g., <candidate>, <running>), which merely mean making immutable nodes visible/invisible in the datastore.

8. NACM Interactions

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.

9. YANG Module

This module imports definitions from [RFC7952], [RFC8342], [RFC8526], and [I-D.ietf-netmod-system-config].

<CODE BEGINS> file "ietf-immutable-annotation@2026-01-12.yang"

```
module ietf-immutable-annotation {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-immutable-annotation";
  prefix imma;

  import ietf-yang-metadata {
    prefix md;
    reference
      "RFC 7952: Defining and Using Metadata with YANG";
  }
  import ietf-netconf-nmda {
    prefix ncds;
    reference
      "RFC 8526: NETCONF Extensions to Support the Network
```

```
         Management Datastore Architecture";
    }
    import ietf-system-datastore {
      prefix sysds;
      reference
        "RFC YYYY: System-defined Configuration";
    }
    import ietf-datastores {
      prefix ds;
      reference
        "RFC 8342: Network Management Datastore Architecture
                   (NMDA)";
    }

    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 called 'immutable'
       to allow the server to formally document existing behavior on
       the mutability of some system configuration. Clients may use
       'immutable' metadata annotation provided by the server to know
       beforehand why certain otherwise valid configuration requests
       will cause the server to return an error.

       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 2026-01-12 {
       description
         "Initial revision.";
       // RFC Ed.: replace XXXX and remove this comment
       reference
         "RFC XXXX: YANG Metadata Annotation for Immutable Flag";
     }
     md:annotation immutable {
       type boolean;
       description
         "The 'immutable' metadata annotation indicates the
          immutability of an instantiated data node. It takes as a
          value 'true' or 'false'. An immutable node cannot be changed
          via configuring a different value in read-write configuration
          datastores (e.g., <running>), though it can be created/deleted
          in read-write configuration datastores. If not specified for
          a given configuration data node, the immutability is the
          same as the value of its parent node in the data tree. The
          default value of 'immutable' annotation for a top-level
          instance node is false if not specified.";
     }

     augment "/ncds:get-data/ncds:input" {
       description
         "Allows the server to include 'immutable' metadata
          annotations in its response to get-data operation.";
       leaf with-immutability {
         when
           "derived-from-or-self(../ncds:datastore,'sysds:system') "
         + "or derived-from-or-self(../ncds:datastore,'ds:intended') "
         + "or derived-from-or-self(../ncds:datastore,'ds:operational')";
         type empty;
         description
           "If this parameter is present, the server returns the
            'immutable' annotation for configuration that it
            internally thinks immutable.";
       }
     }
   }
   <CODE ENDS>
```

10. Security Considerations

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

The "ietf-immutable-annotation" YANG module defines a data model that is designed to be accessed via YANG-based management protocols, such as NETCONF [RFC6241] or RESTCONF [RFC8040]. These protocols have to use a secure transport layer (e.g., SSH [RFC4252], TLS [RFC8446], 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 specified in this document defines a metadata annotation, it also extends the RPC operations of the NETCONF protocol in [RFC6241] and [RFC8526].

The immutable metadata annotation exposes the immutability of configuration data, which may provide hints for attackers to find vulnerabilities in the network, e.g., to leverage the immutability of some configuration to better craft an attack. Since immutable annotations are attached to the instances of configuration data nodes, it is only accessible to clients that have the permissions to read the annotated configuration nodes.

The security considerations for the NETCONF protocol operations (see Section 9 of [RFC6241] and Section 6 of [RFC8526]) also apply to the operations extended in this document.

11. IANA Considerations

11.1. 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-annotation
Registrant Contact: The IESG.
XML: N/A, the requested URIs are XML namespaces.

11.2. The "YANG Module Names" Registry

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

name: ietf-immutable-annotation
prefix: imma
namespace: urn:ietf:params:xml:ns:yang:ietf-immutable-annotation
RFC: XXXX

11.3. RESTCONF Capability URN Registry

This document defines the following capability identifier URNs in the "RESTCONF Capability URNs" registry defined in [RFC8040]:

Index Capability Identifier
:with-immutability urn:ietf:params:restconf:capability:with-immutability:1.0

12. References

12.1. Normative 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-17, 8 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-system-config-17>.

[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/rfc/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/rfc/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/rfc/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/rfc/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/rfc/rfc7950>.

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

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/rfc/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/rfc/rfc8174>.

[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/rfc/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/rfc/rfc8342>.

[RFC8526] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "NETCONF Extensions to Support the Network Management Datastore Architecture", RFC 8526, DOI 10.17487/RFC8526, March 2019, <https://www.rfc-editor.org/rfc/rfc8526>.

[RFC8527] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "RESTCONF Extensions to Support the Network Management Datastore Architecture", RFC 8527, DOI 10.17487/RFC8527, March 2019, <https://www.rfc-editor.org/rfc/rfc8527>.

12.2. Informative References

[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>.

[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/rfc/rfc4252>.

[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/rfc/rfc8340>.

[RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <https://www.rfc-editor.org/rfc/rfc8343>.

[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/rfc/rfc8446>.

[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/rfc/rfc8530>.

[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/rfc/rfc9000>.

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

[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 immutable configuration. 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. For example,

* 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" annotation making it unchangeable. After this the 'interface-timer' shall be defined as a leaf-ref pointing at the 'supported-timer-values'.

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

[RFC8343] defines a YANG data model for the management of network interfaces. When a system-controlled interface is physically present, the system creates an interface entry with valid name and type values in <system> (if exists, see [I-D.ietf-netmod-system-config]).

The system-generated type value is dependent on and represents the hardware present, and as a consequence cannot be changed by the client. If a client tries to set the type of an interface to a value that can never be used by the system, the request will be rejected by the server. The data is modeled as "config true" and thus should be annotated as immutable.

Seemingly an alternative would be to model the list and these leafs 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.

A.3. UC3 - Predefined Administrator Roles

User and group management is fundamental for setting up access control rules (see Section 2.5 of [RFC8341]).

A device may provide a predefined user account (e.g., a system administrator that is always available and has full privileges) for initial system set up and management of other users/groups. It is possible that a new user/group can be defined granted particular privileges, but the predefined administrator account and its granted access are immutable.

A.4. UC4 - Declaring immutable system configuration from the perspective of a logical network element (LNE)

A logical network element (LNE), as described in [RFC8530], is an independently managed virtual network device made up of resources allocated to it from its host or parent network device. 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. Examples of Server's Immutable Behavior

This section provides some examples to illustrate the server's behavior with immutable flag. These examples are not intended as recommendations for real-world deployments. The following fictional module is used throughout this section:

```
   module example-user-group {
     yang-version 1.1;
     namespace "urn:example:user-group";
     prefix "ex-urp";

     import iana-crypt-hash {
       prefix ianach;
     }

     container user-groups {
       list group {
         key "name";
         leaf name {
           type string;
         }
         leaf description {
           type string;
         }
         leaf access-level {
           type enumeration {
             enum admin;
             enum power;
             enum normal;
             enum guest;
           }
         }
         list user {
           key "name";
           leaf name {
             type string;
           }
           leaf password {
             type ianach:crypt-hash;
           }
           leaf full-name {
             type string;
           }
         }
         leaf-list tag {
           type string;
           ordered-by user;
         }
       }
     }
   }
```

B.1. NETCONF Example to Retrieve Immutable Configuration

Figure 2 illustrates a NETCONF request example to retrieve "user-groups" configuration in <system> with "with-immutability" parameter and the response a server might return.

```
================ NOTE: '\' line wrapping per RFC 8792 ================

<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>
      <user-groups xmlns="urn:example:user-group"/>
    </subtree-filter>
    <with-immutability/>
  </get-data>
</rpc>

<rpc-reply message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-nmda">
    <user-groups xmlns="urn:example:user-group"
      xmlns:imma="urn:ietf:params:xml:ns:yang:ietf-immutable-\
                                                        annotation"
      imma:immutable="false">
      <group imma:immutable="true">
        <name>administrator</name>
        <description imma:immutable="false">administrator group</\
                                                      description>
        <access-level>admin</access-level>
        <user>
          <name>ex-username-1</name>
          <password>$5$rounds=10000$mysalt123456789$l4BjA1p/8q.qCYJ.\
                          2pLqjR5mCJf2bP7cLpYWmnC7Hq8</password>
        </user>
        <user imma:immutable="false">
          <name>ex-username-2</name>
          <password>$1$/h1234q$abcdef1234567890abcdef</password>
        </user>
        <tag>system</tag>
        <tag>non-editable</tag>
      </group>
      <group imma:immutable="false">
        <name>power-users</name>
        <description>Power user group</description>
```

```
          <access-level>power</access-level>
          <user>
            <name>ex-username-3</name>
            <password>$1$/h4567q$abcdef2345678901abcdef</password>
          </user>
          <tag>system</tag>
          <tag>editable</tag>
        </group>
      </user-groups>
    </data>
  </rpc-reply>
```

Figure 2: A NETCONF Example to Retrieve Immutable Configuration

B.2. RESTCONF Example to Retrieve Immutable Configuration

Figure 3 illustrates a RESTCONF request example to retrieve "user-groups" configuration in <system> with "with-immutability" query parameter and the response a server might return.

```
================ NOTE: '\' line wrapping per RFC 8792 ================

GET /restconf/ds/ietf-system-datastore:system/example-user-group:\
                                  user-groups?with-immutability HTTP/1.1
Host: example.com
Accept: application/yang-data+json


HTTP/1.1 200 OK
Date: Fri, 9 Jan 2026 15:56:30 GMT
Server: example-server
Content-Type: application/yang-data+json
Cache-Control: no-cache
ETag: "a74eefc993a2b"
Last-Modified: Mon, 5 Jan 2026 14:02:14 GMT

{
  "example-user-group:user-groups": {
    "@": {
      "ietf-immutable-annotation:immutable": false
    },
    "group": [
      {
        "@": {
          "ietf-immutable-annotation:immutable": true
        },
        "name": "administrator",
        "description": "administrator group",
```

```
          "@description": {
            "ietf-immutable-annotation:immutable": false
          },
          "access-level": "admin",
          "user": [
            {
              "name": "ex-username-1",
              "password": "$5$rounds=10000$mysalt123456789$l4BjA1p/8q.\
                                    qCYJ.2pLqjR5mCJf2bP7cLpYWmnC7Hq8"
            },
            {
              "@": {
                "ietf-immutable-annotation:immutable": false
              },
              "name": "ex-username-2",
              "password": "$1$/h1234q$abcdef1234567890abcdef"
            }
          ],
          "tag": ["system", "non-editable"]
        },
        {
          "@": {
            "ietf-immutable-annotation:immutable": false
          },
          "name": "power-users",
          "description": "Power user group",
          "access-level": "power",
          "user": [
            {
              "name": "ex-username-3",
              "password": "$1$/h4567q$abcdef2345678901abcdef"
            }
          ],
          "tag": ["system", "editable"]
        }
      ]
    }
  }
```

Figure 3: A RESTCONF Example to Retrieve Immutable Configuration

B.3. The Inheritance of Immutability

In the example in Figure 2 and Figure 3, there are two "group" list entries inside "user-groups" container node. The "immutable" metadata attribute for "user-groups" container instance is "false", which is also its default value as the top-level element, and thus can be omitted. The "administrator" list entry is immutable with the immutability of its descendant nodes "description" and "user" list entry of "ex-username-2" being explicitly toggled. Other descendant nodes inside "administrator" list entry inherit the immutability of the list entry thus are also immutable.

The "immutable" metadata attribute for "power-users" list entry is "false", which is also the same value as its parent node (i.e., the "user-groups" container), and thus can be omitted. Other descendant nodes inside "power-users" group inherit the immutability of the list entry thus are also mutable.

B.4. Immutability of the list

In the example in Figure 2 and Figure 3, the "group" list as a whole inherits immutability from the container "user-groups", which is mutable. One of the list entry named "administrator" is immutable, and the other entry named "power-user" is mutable. The client is able to copy the entire "user-groups" container in <running>, add new "group" entries, modify the values of descendant nodes of "power-users" list entry, but the values of descendant nodes of "administrator" list entry cannot be overridden with different values expect for the "description" and "ex-username-2" user list entry nodes, which is explicitly reset to be mutable. The client may also subsequently delete any copied "group" entries or the entire "user-groups" container, which will not prevent the deleted data being present in <intended> (if implemented) assuming it is still contained in <system>.

The "user" list inside the "administrator" group list entry as a whole inherits immutability from the list entry, which is immutable. Thus the client cannot add new user entries inside "administrator" group. As one of the user entry named "ex-username-1" is immutable through inheritance, and the other "ex-username-2" user entry is explicitly set to be mutable. The client cannot modify the "password" parameter, or add a "full-name" value for user "ex-username-1". but is allowed to update (e.g., modify the "password" value, or add a "full-name" value) the list entry for user "ex-username-2". The client may copy or subsequently delete any of the two list entries in <running>, but there is no way to delete the nodes from <intended> (if implemented).

B.5. Immutability of the leaf-list

In the example in Figure 2 and Figure 3, the user-ordered "tag" leaf-list node inside the "administrator" group entry as a whole inherits immutability from the list entry, which is immutable. Thus the client cannot add, modify, or reorder entries, the client may copy or subsequently delete any of the two leaf-list entries in <running>, but there is no way to delete the nodes from <intended> if those entries appear in <system>.

The leaf-list node instance inside the "power-users" group entry as a whole inherits immutability from the list entry, which is mutable. Thus the client can add or reorder entries, the client may copy or subsequently delete any of the two leaf-list entries in <running>, but there is no way to delete the nodes from <intended> if those entries appear in <system>.

B.6. Error Response to Clients Overriding Immutable Configuration

Figure 4 provides examples of an attempt to override immutable configuration and the error response that the server might return.

```
   =============== NOTE: '\' line wrapping per RFC 8792 ================

   <rpc message-id="102"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <edit-config>
       <target>
         <running/>
       </target>
       <config>
         <user-groups xmlns="urn:example:user-group">
           <group>
             <name>administrator</name>
             <access-level>guest</access-level>
           </group>
         </user-groups>
       </config>
     </edit-config>
   </rpc>

   <rpc-reply message-id="102"
          xmlns="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="urn:example:user-group">
         /user-groups/group[name="administrator"]/access-level
       </error-path>
       <error-message xml:lang="en">
         Invalid access-level value due to the target node is marked \
                                                              as immutable
       </error-message>
     </rpc-error>
   </rpc-reply>
```

Figure 4: An Example to Override Immutable Configuration with Error Response

Appendix C. Existing Implementations

Note to the RFC Editor: Please remove this section prior to publication.

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

Acknowledgments

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

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

Balazs Lengyel (editor)
Ericsson
Email: balazs.lengyel@ericsson.com

Hongwei Li
HPE
Email: flycoolman@gmail.com