Workgroup: NETMOD
Internet-Draft:
draft-ma-netmod-immutable-flag-06
Published: 28 March 2023
Intended Status: Standards Track
Expires: 29 September 2023
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 as a YANG extension or YANG metadata an existing model handling behavior: modification restrictions on data declared as configuration.

This document defines a YANG extension named "immutable" to indicate that specific "config true" data nodes are not allowed to be created/deleted/updated. To indicate that specific entries of a list/leaf-list node or instances inside list entries cannot be updated/deleted 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 29 September 2023.

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 which 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 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 schema 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;

Clients believe that "config true" nodes are modifiable even though the server is allowed to reject such a modification at any time. If the server knows that it will reject the modification, it should document this towards the clients in a machine readable way.

To address this issue, this document defines a YANG extension named "immutable" to indicate that specific "config true" data nodes are not allowed to be created/deleted/updated. To indicate that specific entries of a list/leaf-list node or instances inside list entries cannot be updated/deleted after initialization, a metadata annotation [RFC7952] 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). Marking instance data nodes as immutable (as opposed to marking schema-nodes) is useful when only some instances of a list or leaf-list shall be marked as read-only.

Immutability is an existing model handling practice. While in some cases it is 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 System defined configuration unchangeable

UC5 Immutable BGP peer type

UC6 Modeling existing data handling behavior in other standard organizations

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] and [RFC8341] and are not redefined here:

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

The following terms are defined in this document:

immutable: A schema or instance node property indicating that the configuration data is not allowed to be created/deleted/updated.

1.2. Applicability

The "immutable" concept defined in this document only indicates write access restrictions to writable datastores. A particular data node or instance MUST have the same immutability in all writable datastores. The immutable annotation information should also be visible 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.

The immutability property of a particular data node or instance MUST be protocol-independent and user-independent.

2. Solution Overview

Already some servers handle immutable configuration data and will reject any attempt to the "create", "delete" or "update" such data. This document allows the existing immutable data node or instance to be marked by YANG extension or metadata annotation. Requests to create/update/delete an immutable configuration data always return an error (if no corresponding "exceptions" are declared in a YANG extension). 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.

However, the following operations SHOULD be allowed for immutable nodes:

* Use a create, update, delete/remove operation on an immutable node 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 with a same value that already exists in the <system> datastore.;

Note that even if a particular data node is immutable without the exception for "delete", it still can be deleted if its parent node is deleted, e.g., /if:interfaces/if:interface/if:type leaf is immutable, but the deletion to the /if:interfaces/if:interface list entry is allowed; if a particular data node is immutable without the exception for "create", it means the client can never create the instance of it, regardless the handling of its parent node; it may be created by the system or have a default value when its parent is created.

In some cases adding the immutable property is allowed but does not have any additional semantic meaning. For example, a key leaf is given a value when a list entry is created, and cannot be modified and deleted unless the list entry is deleted. A mandatory leaf MUST exist and cannot be deleted if the ancestor node exists in the data tree.

3. "Immutable" YANG Extension

3.1. Definition

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 substatment to a "config false" node, but can be allowed anyway. When present, it indicates that data nodes based on the parent statement are not allowed to be added, removed or updated except according to the exceptions argument. Any such write attempt will be rejected by the server.

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.

If more than one value is used, a space-separated string for the "exceptions" argument is used. For example, if a particular data node can be created and modified, but cannot be deleted, the following "immutable" YANG extension with "create" and "update" exceptions should be defined in a substatement to that data node:

```
immutable "create update";
```

Providing an empty string for the "exceptions" argument is equivalent to a single extension without an argument followed. Providing all 3 values can be used to override immutability inherited from its ancestor node. For data nodes with no write access restriction inherited from its ancestor node (see Section 3.2), providing all 3 values has the same effect as not using this extension at all, but can be used anyway.

Note that leaf-list instances can be created and deleted, but not modified. Any exception for "update" operation to leaf-list data nodes SHALL be ignored.

3.2. Inheritance of Immutable YANG Extension

Immutability specified by the use of the 'immutable' extension statement (including any exception argument) is inherited by all child and descendant nodes of a container or a list. It is possible to override thhe inherited immutability property by placing another immuable extension statement on a specific child/descendant node. For example, given the following list definition:

```
list application {
  im:immutable "create delete";
  key name;
  leaf name {
    type string;
  }
  leaf protocol {
    im:immutable;
    type enumeration {
      enum tcp;
      enum udp;
    }
  }
  leaf port-number {
    im:immutable "create update delete";
    type int16;
  }
}
```

application list entries are allowed to be created and deleted, but cannot be modified; "protocol" cannot be changed in any way while "port-number" can be created, modified or deleted. Using the immutable statement with exception argument we can make immutability stricter (for the protocol child node) or less restrictive (for the port-number child node).

4. "Immutable" Metadata Annotation

4.1. Definition

The "immutable" flag SHALL be used to indicate the immutability of a particular instantiated data node. It can only be used for list/leaf-list entries. The "immutable" flag is of type boolean.

Note that "immutable" metadata annotation is used to annotate instances of a list/leaf-list rather than schema nodes. 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.

Any list/leaf-list instance annotated with immutable="true" by the server is read-only to clients and cannot be updated/deleted. If a list entry is annotated with immutable="true", the whole instance is read-only and any contained descendant configuration is not allowed to be created, updated and deleted. Descendant nodes SHALL NOT carry the immutable annotation.

When the client retrieves data from a particular datastore, immutable data node instances MUST be annotated with

immutable="true" by the server. If the "immutable" metadata annotation for a list/leaf-list entry is not specified, the default "immutable" value is false. Explicitly annotating instances as immutable="false" has the same effect as not specifying this value.

5. Interaction between Immutable YANG Extension and Metadata Annotation

When a client reads data from a datastore, if a data node is specified as immutable using the extension statement, the corresponding data node instances generally SHALL NOT be marked with the immutable annotation. However, if the immutable extension statement has exceptions defined, the server MAY decide that for a particular list entriy or leaf-list instance strict immutability shall apply without exceptions. In this case the server SHALL mark the relevant data node instances with the immutable annotation. The immutable annotation overrides any exceptions specified for the immutabile statement inlcuding any exception on any descendant nodes.

6. Interaction between Immutable Flag and NACM

If a data node or some list or leaf-list entries are immutable the server MUST reject any operation that attempts to create, delete or update them, however the "exceptions" argument, if present, SHALL be taken into account. Rejecting an operation due to immutability SHALL be done indepent of any access control settings.

7. YANG Module

```
<CODE BEGINS> file="ietf-immutable@2022-12-14.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;
    }

    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 on the parent data-definition
       statement cannot be created, 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 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 2022-12-14 {
  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 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 space-separated operations that
     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 data node
```

```
        - update: allows users to modify instances of the data node
        - delete: allows users to delete instances of the data node

        To disallow all user write access, omit the argument;

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

        Equivalent YANG definition for this extension:

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

        Immutability specified by the use of the 'immutable' extension
        statement (including any exception argument) is inherited by al
        child and descendant nodes of a container or a list. It is poss
        to override the inherited immutability property by placing anot
        immutable extension statement on a specific child/descendant no

        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. If a list entry is annotated
         with immutable='true', the whole instance is read-only and
         including any contained descendant data nodes.
         The default is 'immutable=false' if not specified for an instan
    }
  }
```

<CODE ENDS>

8. IANA Considerations

8.1. 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.

8.2. 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

9. 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>.

[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>.

[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-01, 4 January 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-system-config-01>.

[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>.

[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 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 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 "create delete";
      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 System defined configuration unchangeable

As stated in [I-D.ietf-netmod-system-config] the device itself might supply some configuration. As defined in that document in section "5.4. Modifying (overriding) System Configuration" the server may allow some parts of system configuration to be modified while other parts of the system configuration are non-modifiable. The immutable extension or metadata annotation can be used to define which parts are non-modifiable and to inform the client about this fact.

A.5. UC5 - Immutable BGP peer type

Another example is the type attribute of BGP neighbors. The peer type of the BGP neighbor is closely related to the network topology: external BGP (EBGP) peer type relationships are established between BGP routers running in different ASs; while internal BGP (IBGP) peer type relationships are established between BGP routers running in the same AS. Thus BGP peer type cannot be changed to the value which does not match the actual one. Since there are EBGP/IBGP-specific configurations which need to reference the "peer-type" node (e.g., in "when" statement) and be conditionally available, it can only be modelled as "config true" but immutable.

Following is the fragment of a simplified BGP module with the /bgp/neighbor/peer-type defined as immutable:

```
container bgp {
  leaf as {
    type inet:as-number;
    mandatory true;
    description
      "Local autonomous system number of the router.";
  }
  list neighbor {
    key "remote-address";
    leaf remote-address {
      type inet:ip-address;
      description
        "The remote IP address of this entry's BGP peer.";
    }
    leaf peer-type {
      im:immutable "create delete";
      type enumeration {
        enum ebgp {
         description
           "External (EBGP) peer.";
        }
        enum ibgp {
          description
            "Internal (IBGP) peer.";
        }
      }
      mandatory true;
      description
        "Specify the type of peering session associated with this
         neighbor. The value can be IBGP or EBGP.";
    }
    leaf ebgp-max-hop {
      when "../peer-type='ebgp'";
      type uint32 {
        range "1..255";
      }
      description
        "The maximum number of hops when establishing an EBGP peer
         relationship with a peer on an indirectly-connected network.
         By default, an EBGP connection can be set up only on a
         directly-connected physical link.";
    }
  }
}
```

A.6. UC6 - Modeling existing data handling behavior in other standard organizations

A number of standard organizations and industry groups (ITU-T, 3GPP, ORAN) already use concepts similar to immutability. These modeling concepts sometimes go back to more than 10 years and cannot be and will not be changed irrespective of the YANG RFCs. Some of these organizations are introducing YANG modelling. Without a formal YANG statement to define data nodes immutable the property is only defined in plain English text in the description statement. The immutable extension and/or metadata annotation can be used to define these existing model properties in a machine-readable way.

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)

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

*Can we do better about the "immutable" terminology?

*Is a Boolean type for immutable metadata annotation sufficient?

*Can immutable data be removed due to a when or choice statement?

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