

NETMOD Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: March 22, 2019

M. Wang  
Q. Wu  
Huawei  
A. Wang  
China Telecom  
September 18, 2018

A YANG Data Model for module revision management  
draft-wang-netmod-module-revision-management-01

## Abstract

This document defines a YANG Data Model for module revision change management. It is intended this model be used by vendors who support multiple revisions of the same YANG module in their systems but implement only one revision of a module. In addition, this document introduces a new generic mechanism based on RPC, denoted as module-revision-change, that allow datanode backwards compatibility detection and provide a report on change type and change details of a YANG module with two or multiple revisions that is defined in design time., e.g., identifies a place in the node hierarchy where data node gets changed or new data gets inserted and indicate whether the change to the data node is backward compatible.

## 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 March 22, 2019.

## Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

<a href="#">1.</a>	Introduction . . . . .	<a href="#">2</a>
<a href="#">2.</a>	Terminologies . . . . .	<a href="#">3</a>
<a href="#">3.</a>	Design of YANG data model for module revision management . .	<a href="#">4</a>
<a href="#">3.1.</a>	yang-modules . . . . .	<a href="#">6</a>
<a href="#">3.1.1.</a>	yang-modules/module . . . . .	<a href="#">6</a>
<a href="#">3.1.2.</a>	yang-modules/change-log . . . . .	<a href="#">6</a>
<a href="#">3.2.</a>	RPC definition for module revision change . . . . .	<a href="#">6</a>
<a href="#">3.2.1.</a>	Usage Example . . . . .	<a href="#">6</a>
<a href="#">4.</a>	YANG Extension for Purpose Marking . . . . .	<a href="#">8</a>
<a href="#">4.1.</a>	Usage Example: Add New Function . . . . .	<a href="#">8</a>
<a href="#">4.2.</a>	Usage example: Bug Fix . . . . .	<a href="#">10</a>
<a href="#">5.</a>	Library Augmentation . . . . .	<a href="#">12</a>
<a href="#">6.</a>	Multiple revisions module management . . . . .	<a href="#">13</a>
<a href="#">7.</a>	Yang Data Model Definition . . . . .	<a href="#">13</a>
<a href="#">8.</a>	Security Considerations . . . . .	<a href="#">21</a>
<a href="#">9.</a>	IANA Considerations . . . . .	<a href="#">21</a>
<a href="#">10.</a>	Acknowledgements . . . . .	<a href="#">22</a>
<a href="#">11.</a>	Normative References . . . . .	<a href="#">22</a>
<a href="#">Appendix A.</a>	Example of Module Revision Management . . . . .	<a href="#">23</a>
	Authors' Addresses . . . . .	<a href="#">25</a>

## [1.](#) Introduction

The revised Network Management Datastore Architecture (NMDA) defines a set of new datastores that each hold YANG-defined data [[RFC7950](#)] and represent a different "viewpoint" on the data that is maintained by a server. To support the NMDA, many modules may need to be updated or restructured especially for some published non-NMDA modules, the model should be republished with an NMDA-compatible structure, deprecating non-NMDA constructs [[I-D.ietf-netmod-rfc6087bis](#)]. Therefore, how to support backward-

compatible and indicate the module's changes is an issue.

As described in [\[RFC7950\]](#) , a module name MUST NOT be changed when definitions contained in a module are available to be imported by any other module and are referenced in "import" statements via the module

name. In some case, when we make non-backward compatible updates, the module name might be forced to change, according to [\[RFC7950\]](#) module update rule.

In order to provide an easy way to indicate how backward-compatible a given YANG module actually is, this document defines a YANG Data Model for module revision change management. It is intended this model be used by vendors who support multiple revisions of the same YANG module in their systems but implement only one revision of a module. In addition, this document introduces a new generic mechanism based on RPC, denoted as module-revision-change, that allow Datanode backwards compatibility detection and provide a report on change type and change details of a YANG module with two or multiple revisions that is defined in design time., e.g., identifies a place in the node hierarchy where data node gets changed or new data gets inserted and indicate whether the change to the data node is backward compatible.

In addition, in order to identify whether changes between two revisions represent bug fixes, new functionality, or both, etc [\[I-D.verdt-netmod-yang-versioning-reqs\]](#), this document also defines a new YANG extension statement which can help the user to understand the purpose of changing a specific node. See More details in [section 4](#).

## [2](#). Terminologies

The keywords "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](#), [\[RFC2119\]](#).

The following terms are defined in [\[RFC6241\]](#) and are not redefined here:

- o client

- o notification
- o server

The following terms are defined in [[RFC7950](#)] and are not redefined here:

action

container

list

operation

### [3.](#) Design of YANG data model for module revision management

The "ietf-module-revision-management" module provides per revision the module change type and change details used by a server. This module is defined using YANG version 1.1, but it supports the description of YANG modules written in any revision of YANG.

All data nodes in "ietf-module-revision-management" are "config false", and thus accessible in the operational state datastore.

Following is the YANG Tree Diagram for the "ietf-module-revision-management" module:

```
module: ietf-module-revision
```

```
  +--ro yang-modules
```

```
    +--ro module* [name revision]
```

```
      +--ro name                yang:yang-identifier
```

```
      +--ro revision            yanglib:revision-identifier
```

```
      +--ro backward-compatible? boolean
```

```
      +--ro revision-change-log* [index]
```

```
        +--ro index            uint32
```

```
        +--ro change-operation  identityref
```

```
        +--ro (yang-statements)?
```

```
          +--:(data-definition-statement)
```

```
            | +--ro data-definition
```

```
            |   +--ro target-node        yang:xpath1.0
```

```
            |   +--ro location-point?    yang:xpath1.0
```

```
            |   +--ro where?             enumeration
```

```
            |   +--ro data-definition?
```

```
          +--:(other-statement)
```

```
            +--ro other-statement
```

```
              +--ro statement-name?      identityref
```

```
              +--ro statement-definition?
```

```
              +--ro substatements* [statement-name]
```

```
                +--ro statement-name      identityref
```

```

        +--ro substatement-definition?
augment /yanglib:yang-library/yanglib:module-set/yanglib:module:
  +--ro backward-compatible?    boolean
augment /yanglib:yang-library/yanglib:module-set/yanglib:module/yanglib:submo
  +--ro backward-compatible?    boolean

rpcs:
  +---x module-revision-change
    +---w input
      | +---w source module-name yang:yang-identifier
      | +---w source-revision?   yanglib:revision-identifier
      | +---w target module-name yang:yang-identifier
      | +---w target-revision?   yanglib:revision-identifier
    +--ro output
      +--ro (status-response)?
      | +--:(wrong-match)
      | | +--ro wrong-match?      empty
      | +--ro data-nodes* [data-node-name]
      |   +--ro data-node-name      string
      |   +--ro is-new-node?        boolean
      |   +--ro change-operation?   identityref

```

### [3.1.](#) yang-modules

This container holds all per revision the module discrepancy information used by a server.

#### [3.1.1.](#) yang-modules/module

This mandatory list contains one entry for each revision of each YANG module that is used by the server. It is possible for multiple revisions of the same module to be imported, in addition to an entry for the revision that is implemented by the server. Multiple revisions of the same module are either backward-compatible or non backward-compatible.

#### [3.1.2.](#) yang-modules/change-log

This list contains one entry for each schema node change from previous revision known by the server, and identifies schema node change path, location, operation and associated with corresponding schema node in the "change-log" list. Each revision of the YANG module has multiple entries.

A change log is an ordered collection of changes that are applied to one revision of YANG module. Each change is identified by an "index", and it has an change operation ("create", "delete", "move", "modify") that is applied to the target resource. Each change can be applied to a sub-resource "target" within the target resource. If the operation is "move", then the "where" parameter indicates how the node is moved. For values "before" and "after", the "point" parameter specifies the data node insertion point.

Each entry within a change log MUST identify exactly one data definition change or other statement change.

### [3.2.](#) RPC definition for module revision change

The "module-revision-change" rpc statement is defined to retrieve the schema data node changes between any two revisions of the same module, i.e. the data node that get updated or newly added during module revision change. This rpc statement takes module identification information as input, and provides the list of data nodes that make changes or are newly added in the later revision.

#### [3.2.1.](#) Usage Example

For example, there are two revisions of the same module, the yang codes are shown as below:

```
module example-a{
  yang-version 1.1;
  namespace "urn:example:a";
  prefix "a";

  organization "foo.";
  contact "fo@example.com";
  description
```

```

        "foo.";

    revision 2017-12-01 {
        description "Initial revision.";
    }

    container system {
        leaf host-name {
            type string;
            description
                "Hostname for this system.";
        }
    }
}

module example-a{
    yang-version 1.1;
    namespace "urn:example:a";
    prefix "a";

    organization "foo.";
    contact "fo@example.com";
    description
        "foo.";

    revision 2017-12-20{
        description "Initial revision.";
    }

    container system {
        leaf host-name {
            type string;
            description
                "Hostname for this system.";
        }
        leaf b {
            type string;
            description
                "foo";
        }
    }
}

```



If we initiate a "module-revision-change" RPC to retrieve the changes between two revisions of module "a", the NETCONF XML example are shown as below:

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <module-revision-change xmlns="http://example.com/system">
    <module-name>example-a</module-name>
    <source-revisions>1.0.0</source-revisions>
    <target-revisions>2.0.0</target-revisions>
  </module-revision-change>
</rpc>

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data-nodes>
    <id>0001</id>
    <data-node-name>b</data-node-name>
    <is-new-node>true</is-new-node>
    <data-node-change>major-change</data-node-change>
  </data-nodes>
</rpc-reply>
```

#### [4.](#) YANG Extension for Purpose Marking

In order to identify whether changes between two revisions represent bug fixes, new functionality, or both, etc [[I-D.verdt-netmod-yang-versioning-reqs](#)], purpose marking are defined via the YANG extension "mr:purpose". This YANG extension statement is defined in the module "ietf-module-revision" ([Section 7](#)). This YANG extension can help the user to understand the purpose of changing a specific node.

##### [4.1.](#) Usage Example: Add New Function

For example, there is a YANG data model "example-a", the yang codes are shown as below:

```
module example-a{
  yang-version 1.1;
  namespace "urn:example:a";
  prefix "a";

  organization "foo.";
  contact "fo@example.com";
  description
    "foo.";

  revision 2017-12-01 {
    description "Initial revision.";
  }

  container system {
    leaf host-name {
      type string;
      description
        "Hostname for this system.";
    }
  }
}
```

When the author of "example-a" designs a new revision to add some new attributes, the "mr:purpose" marking can be used to mark the purpose of the updates. For example:

```
module example-a{
  yang-version 1.1;
  namespace "urn:example:a";
  prefix "a";

  organization "foo.";
  contact "fo@example.com";
  description
    "foo.";

  revision 2017-12-20{
    description "Initial revision.";
  }

  container system {
    leaf host-name {
      type string;
      description
        "Hostname for this system.";
    }
    leaf b {
      type string;
      mr:purpose "new-function";
      description
        "foo";
    }
  }
}
```

#### [4.2.](#) Usage example: Bug Fix

For example, there are some bugs in a published model "example-b", the yang codes are shown as below:

```
module example-b{
  yang-version 1.1;
  namespace "urn:example:b";
  prefix "b";

  organization "foo.";
  contact "fo@example.com";
  description
    "foo.";

  revision 2017-12-01 {
    description "Initial revision.";
  }

  container system {
    leaf host-name {
      type uint32;
      description
        "Hostname for this system.";
    }
  }
}
```

The following updates allow to fix bugs in a backward-compatible way:

```
module example-b{
  yang-version 1.1;
  namespace "urn:example:b";
  prefix "b";

  organization "foo.";
  contact "fo@example.com";
  description
    "foo.";

  revision 2017-12-01 {
    description "Initial revision.";
  }

  container system {
    leaf host-name-update {
      type string;
      mr:purpose "bug-fix";
      description
        "Hostname for this system.";
    }
    leaf host-name {
      type uint32;
      status deprecated;
      description
```

```

        "Hostname for this system.";
    }
}
}

```

## 5. Library Augmentation

Backward compatibility for each revision of YANG module can also be read using the yang library [[I-D.ietf-netconf-rfc7895bis](#)] if a server supports both YANG library and the augmentation defined below. If a server supports indication of backward compatibility for one revision of and the YANG module, it SHOULD also support the "ietf-module-revision" module.

The tree associated with the defined augmentation is:

```

module: ietf-module-revisions
  augment /yanglib:yang-library/yanglib:modules/yanglib:module:
  +--ro backward-compatible?  bool

augment /yanglib:yanglibrary/yanglib:modules/yanglib:module
/yanglib:submodule:
  +--ro backward-compatible?  bool

```

## 6. Multiple revisions module management

As experience is gained with a module, it may be desirable to support multiple revisions of that module in their systems but implement one revision of a module at each time. To indicate the details changes of that module, e.g., identifies schema node change path, location, operation and associated with corresponding schema node, it will be desirable to use 'ietf-module-revision' defined in this document to manage all the revisions of that module and keep track of module change discrepancy in different revision, especially when the new

revision is not backward compatible with previous revision.

## 7. Yang Data Model Definition

<CODE BEGINS> file "ietf-module-revision@2018-08-08.yang"

```
module ietf-module-revision {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-module-revision";
  prefix ml;

  import ietf-yang-library {
    prefix yanglib;
  }
  import ietf-yang-types {
    prefix yang;
  }

  organization
    "IETF Network Modeling (NETMOD) Working Group";
  contact
    "WG Web:  <https://datatracker.ietf.org/wg/netmod/>

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

    Author:   Qin Wu
              <mailto:bill.wu@huawei.com>
              Zitao Wang
              <mailto:wangzitao@huawei.com>";
  description
```

"This YANG module defines an module log.";

```
revision 2018-08-08 {
  description
    "Initial revision.";
  reference "RFC XXXX: Using Metadata with YANG for Module revisions";
}

identity operation-type {
  description
    "Abstract base identity for the operation type ";
```

```

}

identity create {
    base operation-type;
    description
        "Denotes create new data nodes";
}

identity delete {
    base operation-type;
    description
        "Denotes delete the target node";
}

identity move {
    base operation-type;
    description
        "Denote move the target node.";
}

identity modify {
    base operation-type;
    description
        "Denote modify the target data node.";
}

identity statement-type {
    description
        "Base identity for statement type";
}

identity feature-statement {
    base statement-type;
    description
        "feature statement, if this type be chose, it means that the
        feature or if-feature statement been modified";
}

```

```

identity identity-statement {
    base statement-type;
    description
        "identity statement, if this type be chose, it means that the

```



```

    identity statement been modified, for example, add new identity, etc.";
}

identity grouping-statement {
    base statement-type;
    description
        "grouping statement, if this type be chose, it means that the grouping
        statement been modified.";
}

identity typedef-statement {
    base statement-type;
    description
        "typedef statement, if this type be chose, it means that the typedef
        statement been modified.";
}

identity augment-statement {
    base statement-type;
    description
        "augment statement, if this type be chose, it means that the augment
        statement been modified.";
}

identity rpc-statement {
    base statement-type;
    description
        "rpc statement, if this type be chose, it means that the rpc
        statement been modified.";
}

identity notification-statement {
    base statement-type;
    description
        "notification statement, if this type be chose, it means that the notific
        statement been modified.";
}

extension purpose {
    argument name;
    description
        "The purpose can be used to mark the data nodes change purpose.
        The name argument can be specified in the following recommended mode

```

- bug-fix, which can help user to understand the data nodes' changes pre
- new-function, which can help user to understand the data nodes' change
- nmda-conform, which can help user to understand the data nodes' change

and note that the user can argument the purpose name according to their  
}

```
grouping data-definition {
  container data-definition {
    leaf target-node {
      type yang:xpath1.0;
      mandatory true;
      description
        "Identifies the target data node for update.
        Notice that, if the update-type equal to move or delete,
        this target-node must point to the data node of old version.
        \t
        For example, suppose the target node is a YANG leaf named a,
        and the previous version is:
        \t
        container foo {
          leaf a { type string; }
          leaf b { type int32; }
        }
        \t
        the new version is:
        container foo {
          leaf b {type int32;}
        }
        \t
        Therefore, the targe-node should be /foo/a.";
```

```
    }
    leaf location-point {
      type yang:xpath1.0;
      description
        "Identifies the location point where the updates happened.";
    }
    leaf where {
      when "derived-from-or-self(..../change-operation, 'move')" {
        description
          "This leaf only applies for 'move'
          updates.";
      }
      type enumeration {
        enum "before" {
          description
            "Insert or move a data node before the data resource
```

identified by the 'point' parameter.";

```
    }
    enum "after" {
      description
        "Insert or move a data node after the data resource
        identified by the 'point' parameter.";
    }
    enum "first" {
      description
        "Insert or move a data node so it becomes ordered
        as the first entry.";
    }
    enum "last" {
      description
        "Insert or move a data node so it becomes ordered
        as the last entry.";
    }
  }
  default "last";
  description
    "Identifies where a data resource will be inserted
    or moved.";
}
anydata data-definition {
  when "derived-from-or-self(..../change-operation, 'modify')" {
    description
      "This nodes only be present when
      the 'change-operation' equal to 'modify'.";
  }
  description
    "This nodes used for present the definitions before updated.
    And this nodes only be present when
    the 'change-operation' equal to 'modify'.";
}
  description
    "Container for data statement";
}
description
  "Grouping for data definition";
}
```

```

grouping other-statement {
  container other-statement {
    leaf statement-name {
      type identityref {
        base statement-type;
      }
      description
        "Statement name, for example, identity, feature, typedef, etc.";
    }
  }
}

```

```

}
anydata statement-definition {
  description
    "This nodes used for present new the definitions.";
}
list substatements {
  key "statement-name";
  leaf statement-name {
    type identityref {
      base statement-type;
    }
    description
      "Statement name, for example, identity, feature, typedef, etc.";
  }
  anydata substatement-definition {
    description
      "This nodes used for present new the definitions.";
  }
  description
    "List for substatements updates";
}
description
  "Container for header statement updates";
}
description
  "Grouping for header statement";
}

grouping change-log {
  list revision-change-log {
    key "index";
    leaf index {
      type uint32;
    }
  }
}

```

```

        description
            "Index for module change log";
    }
    leaf change-operation {
        type identityref {
            base operation-type;
        }
        mandatory true;
        description
            "This leaf indicate the change operation, such as create, move, delet
    }
    choice yang-statements {
        description
            "Choice for various YANG statements that have been impacted.";
        case data-definition-statement {

```

```

        uses data-definition;
    }
    case other-statement {
        uses other-statement;
    }
}
description
    "List for module revision change log";
}
description
    "Grouping for module revision change log";
}

container yang-modules {
    config false;
    list module {
        key "name revision";
        leaf name {
            type yang:yang-identifier;
            description
                "The YANG module or submodule name.";
        }
        leaf revision {
            type yanglib:revision-identifier;
            description
                "The YANG module or submodule revision date.  If no revision

```

```

        statement is present in the YANG module or submodule, this
        leaf is not instantiated.";
    }
    leaf backward-compatible {
        type boolean;
        description
            "Indicates whether it is a backward compatible version.
            If this parameter is set to true, it means that this version is
            a backwards compatible version";
    }
    uses change-log;
    description
        "List for module updated log";
    }
    description
        "This container present the modules updated log.";
    }
    augment "/yanglib:yang-library/yanglib:module-set/yanglib:module" {
        description
            "Augment the yang library with backward compatibility indication.";
        leaf backward-compatible {
            type boolean;

```

```

        description
            "backward compatibility indication.";
    }
    }
    augment "/yanglib:yang-library/yanglib:module-set/yanglib:module/yanglib:subm
    description
        "Augment the yang library with backward compatibility indication.";
    leaf backward-compatible {
        type boolean;
        description
            "backward compatibility indication.";
    }
    }
    rpc module-revision-change {
        description
            "Module Node change query operation.";
        input {
            leaf source-module-name {
                type yang:yang-identifier;

```

```

    mandatory true;
    description
        "The Source YANG module or submodule name.";
}
leaf source-revision {
    type yanglib:revision-identifier;
    description
        "The Source YANG module revision date.  If no revision
        statement is present in the YANG module or submodule, this
        leaf is not instantiated.";
}
leaf target-module-name {
    type yang:yang-identifier;
    mandatory true;
    description
        "The Target YANG module or submodule name.";
}
leaf target-revision {
    type yanglib:revision-identifier;
    description
        "The target YANG module revision date.  If no revision
        statement is present in the YANG module or submodule, this
        leaf is not instantiated.";
}
}
output {
    choice status-response{
        leaf wrong-match{
            type empty;

```

```

    description
        "This leaf indicates that two modules have nothing in common.";
}
list data-nodes {
    key "data-node-name";
    description
        "Each entry represents a data node of a given module that
        have been changed from source revision of
        a module to target revision of the module.";
    leaf data-node-name {
        type string;
        description

```





XML: N/A, the requested URI is an XML namespace.

---

This document registers a YANG module in the YANG Module Names registry [[RFC6020](#)].

---

Name:	ietf-module-revision
Namespace:	urn:ietf:params:xml:ns:yang:ietf-module-revision
Prefix:	md
Reference:	RFC xxxx

---

## [10.](#) Acknowledgements

This work is motivated from the discussions of module version in IETF 100 Singapore meeting. Thanks to Juergen Schoenwaelder and Adrian Farrel for useful comments on this work.

## [11.](#) Normative References

[I-D.ietf-netconf-rfc7895bis]

Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", [draft-ietf-netconf-rfc7895bis-04](#) (work in progress), January 2018.

[I-D.ietf-netmod-rfc6087bis]

Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", [draft-ietf-netmod-rfc6087bis-15](#) (work in progress), December 2017.

[I-D.verdt-netmod-yang-versioning-reqs]

Clarke, J., "YANG Module Versioning Requirements", [draft-verdt-netmod-yang-versioning-reqs-00](#) (work in progress), July 2018.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", March 1997.

[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>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", [RFC 7223](#), DOI 10.17487/RFC7223, May 2014, <<https://www.rfc-editor.org/info/rfc7223>>.
- [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>>.

#### [Appendix A](#). Example of Module Revision Management

This section provides an example to generate XML snippet of ietf-module-revision based on the changes between the new revision of interface model as specified in [[draft-ietf-netmod-rfc7223bis](#)] and the old revision of interface model as specified in [[RFC7223](#)].

```
<yang-modules>
  <module>
    <name>ietf-interfaces</name>
    <revision>2018-01-09</revision>
    <backwards-compatible>>false</backwards-compatible>
    <revision-change-log>
      <index>0001</index>
      <change-operation>delete</change-operation>
      <yang-statements>
        <data-definition-statement>
          <data-definition>
            <target>/if:interfaces-state</target>
          </data-definition>
        </data-definition-statement>
      </yang-statements>
    </revision-change-log>

    <revision-change-log>
      <index>0002</index>
      <change-operation>create</change-operation>
      <yang-statements>
        <other-statement>
          <statement>
            <statement-name>feature-statement</statement-name>
```

Internet-Draft

Module Revision Management

September 2018

```
<statement-definition>
  feature if-mib {
    description
      "This feature indicates that the device implements
       the IF-MIB.";
    reference
      "RFC 2863: The Interfaces Group MIB";
  }
</statement-definition>
</statement>
</other-statement>
</yang-statements>
</revision-change-log>

<revision-change-log>
  <index>0003</index>
  <change-operation>modify</change-operation>
  <yang-statements>
    <data-definition-statement>
      <data-definition>
        <target>
          /if:interfaces/if:interface/if:link-up-down-trap-enable
        </target>
        <data-node>
          leaf link-up-down-trap-enable {
            if-feature if-mib; // add if-feature statement
            type enumeration {
              ....
            }
          }
        </data-node>
      </data-definition>
    </data-definition-statement>
  </yang-statements>
</revision-change-log>

<revision-change-log>
  <index>0004</index>
  <change-operation>move</change-operation>
  <yang-statements>
    <data-definition-statement>
      <data-definition>
        <target>
          /if:interfaces-state/if:interface/if:admin-status
```

```

    </target>
    <location-point>
      /if:interfaces/if:interface/link-up-down-trap-enable
    </location-point>
    <where>after</where>
  </data-definition-statement>

```

```

    </yang-statements>
  </revision-change-log>

  <revision-change-log>
    <index>0005</index>
    <change-operation>move</change-operation>
    <yang-statements>
      <data-definition-statement>
        <data-definition>
          <target>
            /if:interfaces-state/if:interface/if:statistics
          </target>
          <location-point>
            /if:interfaces/if:interface/if:speed
          </location-point>
          <where>after</where>
        </data-definition-statement>
      </yang-statements>
    </revision-change-log>
    .....
  </module>
</yang-modules>

```

#### Authors' Addresses

Michael Wang  
 Huawei Technologies, Co., Ltd  
 101 Software Avenue, Yuhua District  
 Nanjing 210012  
 China

Email: wangzitao@huawei.com

Qin Wu

Huawei  
101 Software Avenue, Yuhua District  
Nanjing, Jiangsu 210012  
China

Email: bill.wu@huawei.com

Wang, et al.

Expires March 22, 2019

[Page 25]

---

Internet-Draft

Module Revision Management

September 2018

Aijun Wang  
China Telecom  
Beiqijia Town, Changping District  
Beijing, Beijing 102209  
China

Email: wangaj.bri@chinatelecom.cn

