Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 3, 2015

A. Bierman
YumaWorks
M. Bjorklund
Tail-f Systems
K. Watsen
Juniper Networks
January 30, 2015

YANG Patch Media Type
draft-ietf-netconf-yang-patch-03

Abstract

This document describes a method for applying patches to NETCONF datastores using data defined with the YANG data modeling language.

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 http://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 August 3, 2015.

Table of Contents

1. Introduction

There is a need for standard mechanisms to patch NETCONF [RFC6241] datastores which contain conceptual data that conforms to schema specified with YANG [RFC6020]. An "ordered edit list" approach is needed to provide client developers with a simpler edit request format that can be more efficient and also allow more precise client control of the transaction procedure than existing mechanisms.

This document defines a media type for a YANG-based editing mechanism that can be used with the HTTP PATCH method [RFC5789] or custom NETCONF operations (defined with the YANG rpc-stmt).

YANG Patch is designed to support multiple protocols with the same mechanisms. The RESTCONF protocol defined in [I-D.ietf-netconf-restconf] utilizes YANG Patch with the HTTP PATCH method. A new RPC operation can be defined to utilize YANG Patch in the NETCONF protocol. Both the RESTCONF and NETCONF protocols are designed to utilize the YANG data modeling language to specify content schema modules.

1.1. Terminology

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].

1.1.1. NETCONF

The following terms are defined in [RFC6241]:

- candidate configuration datastore
- client
- configuration data
- datastore
- configuration datastore
- protocol operation
- running configuration datastore
- server
- startup configuration datastore
- state data
- user

1.1.2. HTTP

The following terms are defined in [RFC2616]:

- entity tag
- fragment
- header line
- message body
- method
- path
- query
- request URI
- response body

1.1.3. YANG

The following terms are defined in [RFC6020]:

- container
- data node
- key leaf
- leaf
- leaf-list
- list
- presence container (or P-container)
- RPC operation (now called protocol operation)
- non-presence container (or NP-container)
- ordered-by system
- ordered-by user

1.1.4. RESTCONF

The following terms are defined in [I-D.ietf-netconf-restconf]:

- data resource
- datasource resource
- patch
- RESTCONF capability
- target resource

1.1.5. Terms

The following terms are used within this document:

- YANG Patch: a conceptual edit request using the "yang-patch" YANG container, defined in Section 3. In HTTP, refers to a PATCH method where the media type is "application/yang.patch+xml" or "application/yang.patch+json".
- YANG Patch Status: a conceptual edit status response using the YANG "yang-patch-status" container, defined in Section 3. In HTTP, refers to a response message for a PATCH method, where the message body is identified by the media type "application/yang.patch-status+xml" or "application/yang.patch-status+json".

1.1.6. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

- Brackets "[" and "]" enclose list keys.
- Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- Symbols after data node names: "?" means an optional node and "*" denotes a "list" and "leaf-list".
- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- Ellipsis ("...") stands for contents of subtrees that are not shown.

2. YANG Patch

A "YANG Patch" is an ordered list of edits that are applied to the target datastore by the server. The specific fields are defined with the 'application/yang.patch' extension definition in the YANG module Section 3.

For RESTCONF, the YANG Patch operation is invoked by the client by sending a PATCH method request with the YANG Patch media type. A message body representing the YANG Patch input parameters MUST be provided.

The RESTCONF server MUST return the Accept-Patch header in an OPTIONS response, as specified in [RFC5789], which includes the media type for YANG Patch.

Example:

```
Accept-Patch: application/yang.patch
```

2.1. Target Resource

The YANG Patch operation uses a conceptual root within a NETCONF configuration datastore to identity the patch point for the edit operation. This root can be the datastore itself, or 1 or more data nodes within the datastore.

For RESTCONF, the target resource is derived from the request URI.

For NETCONF, the target resource MUST be defined as an input parameter in the YANG "rpc" statement.

2.2. yang-patch Input

A data element representing the YANG Patch is sent by the client to specify the edit operation request. When used with the HTTP PATCH method, this data is identified by the YANG Patch media type.

YANG Tree Diagram For "yang-patch" Container

```
   +--rw yang-patch
      +--rw patch-id?   string
      +--rw comment?    string
      +--rw edit [edit-id]
         +--rw edit-id      string
         +--rw operation    enumeration
         +--rw target       target-resource-offset
         +--rw point?       target-resource-offset
         +--rw where?       enumeration
         +--rw value
```

2.3. yang-patch-status Output

A data element representing the YANG Patch Status is returned to the client to report the detailed status of the edit operation. When used with the HTTP PATCH method, this data is identified by the YANG Patch Status media type, and the syntax specification is defined by the 'application/yang.patch-status' extension statement defined in Section 3.

YANG Tree Diagram For "yang-patch-status" Container:

```
   +--rw yang-patch-status
      +--rw patch-id?        string
      +--rw (global-status)?
      |  +--:(global-errors)
      |  |  +--ro errors
      |  |
      |  +--:(ok)
      |     +--rw ok?              empty
      +--rw edit-status
         +--rw edit [edit-id]
            +--rw edit-id     string
            +--rw (edit-status-choice)?
               +--:(ok)
               |  +--rw ok?         empty
               +--:(errors)
                  +--ro errors
```

2.4. Target Data Node

The target data node for each edit operation is determined by the value of the target resource in the request and the "target" leaf within each "edit" entry.

If the target resource specified in the request URI identifies a datastore resource, then the path string in the "target" leaf is an

absolute path expression. The first node specified in the "target" leaf is a top-level data node defined within a YANG module.

If the target resource specified in the request URI identifies a data resource, then the path string in the "target" leaf is a relative path expression. The first node specified in the "target" leaf is a child node of the data node associated with the target resource.

2.5. Edit Operations

Each YANG patch edit specifies one edit operation on the target data node. The set of operations is aligned with the NETCONF edit operations, but also includes some new operations.

Operation	Description
create	create a new data resource if it does not already exist or error
delete	delete a data resource if it already exists or error
insert	insert a new user-ordered data resource
merge	merge the edit value with the target data resource; create if it does not already exist
move	re-order the target data resource
replace	replace the target data resource with the edit value
remove	remove a data resource if it already exists or no error

YANG Patch Edit Operations

2.6. Error Handling

If a well-formed, schema-valid YANG Patch message is received, then the server will process the supplied edits in ascending order. The following error modes apply to the processing of this edit list:

All the specified edits MUST be applied or the target datastore contents SHOULD be returned to its original state before the PATCH method started. The server MAY fail to restore the contents of the target datastore completely and with certainty. It is possible for a rollback to fail or an "undo" operation to fail.

The server will save the running datastore to non-volatile storage if it has changed, after the edits have been attempted.

2.7. yang-patch RESTCONF Capability

A URI is defined to identify the YANG Patch extension to the base RESTCONF protocol. If the server supports the YANG Patch media type, then the "yang-patch" RESTCONF capability defined in Section 4.4 MUST be present in the "capability" leaf-list in the "ietf-restconf-monitoring" module defined in [I-D.ietf-netconf-restconf].

3. YANG Module

The "ietf-yang-patch" module defines conceptual definitions with the 'restconf-media-type' extension statements, which are not meant to be implemented as datastore contents by a server.

The "ietf-restconf" module from [I-D.ietf-netconf-restconf] is used by this module for the 'restconf-media-type' extension definition.

RFC Ed.: update the date below with the date of RFC publication and remove this note.

<CODE BEGINS> file "ietf-yang-patch@2015-01-24.yang"

```
module ietf-yang-patch {
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-patch";
  prefix "ypatch";

  import ietf-restconf {
    prefix rc;
    revision-date 2015-01-30;
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:   <http://tools.ietf.org/wg/netconf/>
     WG List:  <mailto:netconf@ietf.org>

     WG Chair: Mehmet Ersue
               <mailto:mehmet.ersue@nsn.com>

     WG Chair: Mahesh Jethanandani
               <mailto:mjethanandani@gmail.com>

     Editor:   Andy Bierman
               <mailto:andy@yumaworks.com>
```

```
        Editor:   Martin Bjorklund
                  <mailto:mbj@tail-f.com>

        Editor:   Kent Watsen
                  <mailto:kwatsen@juniper.net>";

    description
      "This module contains conceptual YANG specifications
       for the YANG Patch and YANG Patch Status data structures.

       Note that the YANG definitions within this module do not
       represent configuration data of any kind.
       The YANG grouping statements provide a normative syntax
       for XML and JSON message encoding purposes.

       Copyright (c) 2015 IETF Trust and the persons identified as
       authors of the code.  All rights reserved.

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

       This version of this YANG module is part of RFC XXXX; see
       the RFC itself for full legal notices.";

    // RFC Ed.: replace XXXX with actual RFC number and remove this
    // note.

    // RFC Ed.: remove this note
    // Note: extracted from draft-ietf-netconf-yang-patch-03.txt

    // RFC Ed.: update the date below with the date of RFC publication
    // and remove this note.
    revision 2015-01-30 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: YANG Patch Media Type.";
    }

    typedef target-resource-offset {
      type string {
        length "1 .. max";
      }
      description
```

```
        "Contains a relative Data Resource Identifier formatted string
         to identify a specific data sub-resource instance.
         The document root for all data resources is a
         target data resource that is specified in the
         object definition using this data type.";
    }

    rc:restconf-media-type "application/yang.patch" {
      uses yang-patch;
    }

    rc:restconf-media-type "application/yang.patch-status" {
      uses yang-patch-status;
    }

    grouping yang-patch {

      description
        "A grouping that contains a YANG container
         representing the syntax and semantics of a
         YANG Patch edit request message.";

      container yang-patch {
        description
          "Represents a conceptual sequence of datastore edits,
           called a patch. Each patch is given a client-assigned
           patch identifier. Each edit MUST be applied
           in ascending order, and all edits MUST be applied.
           If any errors occur, then the target datastore MUST NOT
           be changed by the patch operation.

           A patch MUST be validated by the server to be a
           well-formed message before any of the patch edits
           are validated or attempted.

           YANG datastore validation (defined in RFC 6020, section
           8.3.3) is performed after all edits have been
           individually validated.

           It is possible for a datastore constraint violation to occur
           due to any node in the datastore, including nodes not
           included in the edit list. Any validation errors MUST
           be reported in the reply message.";

        reference
          "RFC 6020, section 8.3.";

        leaf patch-id {
```

```
          type string;
          description
            "An arbitrary string provided by the client to identify
             the entire patch.  This value SHOULD be present in any
             audit logging records generated by the server for the
             patch. Error messages returned by the server pertaining
             to this patch will be identified by this patch-id value.";
        }

        leaf comment {
          type string {
            length "0 .. 1024";
          }
          description
            "An arbitrary string provided by the client to describe
             the entire patch.  This value SHOULD be present in any
             audit logging records generated by the server for the
             patch.";
        }

        list edit {
          key edit-id;
          ordered-by user;

          description
            "Represents one edit within the YANG Patch
             request message.  The edit list is applied
             in the following manner:

               - The first edit is conceptually applied to a copy
                 of the existing target datastore, e.g., the
                 running configuration datastore.
               - Each ascending edit is conceptually applied to
                 the result of the previous edit(s).
               - After all edits have been successfully processed,
                 the result is validated according to YANG constraints.
               - If successful, the server will attempt to apply
                 the result to the target datastore. ";

          leaf edit-id {
            type string;
            description
              "Arbitrary string index for the edit.
               Error messages returned by the server pertaining
               to a specific edit will be identified by this
               value.";
          }
```

```
          leaf operation {
            type enumeration {
              enum create {
                description
                  "The target data node is created using the
                   supplied value, only if it does not already
                   exist.";
              }
              enum delete {
                description
                  "Delete the target node, only if the data resource
                   currently exists, otherwise return an error.";
              }
              enum insert {
                description
                  "Insert the supplied value into a user-ordered
                   list or leaf-list entry. The target node must
                   represent a new data resource.";
              }
              enum merge {
                description
                  "The supplied value is merged with the target data
                   node.";
              }
              enum move {
                description
                  "Move the target node. Reorder a user-ordered
                   list or leaf-list. The target node must represent
                   an existing data resource.";
              }
              enum replace {
                description
                  "The supplied value is used to replace the target
                   data node.";
              }
              enum remove {
                description
                  "Delete the target node if it currently exists.";
              }
            }
            mandatory true;
            description
              "The datastore operation requested for the associated
               edit entry";
          }

          leaf target {
            type target-resource-offset;
```

```
        mandatory true;
        description
          "Identifies the target data resource for the edit
           operation.";
      }

      leaf point {
        when "(../operation = 'insert' or " +
          "../operation = 'move') and " +
          "(../where = 'before' or ../where = 'after')" {
          description
            "Point leaf only applies for insert or move
             operations, before or after an existing entry.";
        }
        type target-resource-offset;
        description
          "The absolute URL path for the data node that is being
           used as the insertion point or move point for the
           target of this edit entry.";
      }

      leaf where {
        when "../operation = 'insert' or ../operation = 'move'" {
          description
            "Where leaf only applies for insert or move
             operations.";
        }
        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. YANG only allows these operations for
               list and leaf-list data nodes that are ordered-by
               user.";
          }

          anyxml value {
            when "(../operation = 'create' or " +
              "../operation = 'merge' " +
              "or ../operation = 'replace' or " +
              "../operation = 'insert')" {
              description
                "Value node only used for create, merge,
                 replace, and insert operations";
            }
            description
              "Value used for this edit operation.
               The anyxml value MUST represent a container with
               exactly one child node, which MUST identify the
               target resource associated with the 'target' leaf.

               For example, suppose the target node is a YANG container
               named foo:

                   container foo {
                     leaf a { type string; }
                     leaf b { type int32; }
                   }

               The value node will contain one instance of foo:

                   <value>
                      <foo xmlns='example-foo-namespace'>
                         <a>some value</a>
                         <b>42</b>
                      </foo>
                   </value>
                ";
          }
        }
      }

    } // grouping yang-patch
```

```
     grouping yang-patch-status {

       description
         "A grouping that contains a YANG container
          representing the syntax and semantics of
          YANG Patch status response message.";

       container yang-patch-status {
         description
           "A container representing the response message
            sent by the server after a YANG Patch edit
            request message has been processed.";

         leaf patch-id {
           type string;
           description
             "The patch-id value used in the request";
         }

         choice global-status {
           description
             "Report global errors or complete success.
              If there is no case selected then errors
              are reported in the edit-status container.";

           case global-errors {
             uses rc:errors;
             description
               "This container will be present if global
                errors unrelated to a specific edit occurred.";
           }
           leaf ok {
             type empty;
             description
               "This leaf will be present if the request succeeded
                and there are no errors reported in the edit-status
                container.";
           }
         }

         container edit-status {
           description
             "This container will be present if there are
              edit-specific status responses to report.
              If all edits succeeded and the 'global-status'
              returned  is 'ok', then a server MAY omit this
              container";
```

```
          list edit {
            key edit-id;

            description
              "Represents a list of status responses,
               corresponding to edits in the YANG Patch
               request message.  If an edit entry was
               skipped or not reached by the server,
               then this list will not contain a corresponding
               entry for that edit.";

            leaf edit-id {
              type string;
               description
                 "Response status is for the edit list entry
                  with this edit-id value.";
            }
            choice edit-status-choice {
              description
                "A choice between different types of status
                 responses for each edit entry.";
              leaf ok {
                type empty;
                description
                  "This edit entry was invoked without any
                   errors detected by the server associated
                   with this edit.";
              }
              case errors {
                uses rc:errors;
                description
                  "The server detected errors associated with the
                    edit identified by the same edit-id value.";
              }
            }
          }
        }
      }  // grouping yang-patch-status

   }

   <CODE ENDS>
```

4. IANA Considerations

4.1. YANG Module Registry

This document registers one URI in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made.

URI: urn:ietf:params:xml:ns:yang:ietf-yang-patch
Registrant Contact: The NETMOD WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

This document registers one YANG module in the YANG Module Names registry [RFC6020].

```
name:         ietf-yang-patch
namespace:    urn:ietf:params:xml:ns:yang:ietf-yang-patch
prefix:       ypatch
// RFC Ed.: replace XXXX with RFC number and remove this note
reference:    RFC XXXX
```

4.2. application/yang.patch Media Types

The MIME media type for a YANG Patch document is application/yang.patch.

Type name: application

Subtype name: yang.patch

Required parameters: TBD

Optional parameters: TBD

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

// RFC Ed.: replace XXXX with RFC number and remove this note
Published specification: RFC XXXX

4.3. application/yang.patch-status Media Types

The MIME media type for a YANG Patch status document is application/yang.patch-status.

Type name: application

Subtype name: yang.patch-status

Required parameters: TBD

Optional parameters: TBD

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

// RFC Ed.: replace XXXX with RFC number and remove this note
Published specification: RFC XXXX

4.4. RESTCONF Capability URNs

This document registers several capability identifiers in "RESTCONF Protocol Capability URNs" registry

Index Capability Identifier
:yang-patch urn:ietf:params:restconf:capability:yang-patch:1.0

5. Security Considerations

The YANG Patch media type does not introduce any significant new security threats, beyond what is described in [I-D.ietf-netconf-restconf]. This document defines edit processing instructions for a variant of the PATCH method, as used within the RESTCONF protocol.

It is important for server implementations to carefully validate all the edit request parameters in some manner. If the entire YANG Patch request cannot be completed, then no configuration changes to the system are done.

A server implementation SHOULD attempt to prevent system disruption due to partial processing of the YANG Patch edit list. It may be possible to construct an attack on such a server, which relies on the edit processing order mandated by YANG Patch.

6. Normative References

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-04 (work in progress), January 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, March 2010.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC7158] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7158, March 2013.

[W3C.REC-xml-20081126]
Yergeau, F., Maler, E., Paoli, J., Sperberg-McQueen, C., and T. Bray, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126>.

Appendix A. Acknowledgements

The authors would like to thank the following people for their contributions to this document: Rex Fernando.

Appendix B. Change Log

-- RFC Ed.: remove this section before publication.

B.1. 02 to 03

- added usage of restconf-media-type extension to map the yang-patch and yang-patch-status groupings to media types
- added yang-patch RESTCONF capability URI
- Added sub-section for terms used from RESTCONF
- filled in security considerations section

B.2. 01 to 02

- Reversed order of change log
- Clarified anyxml structure of "value" parameter within a YANG patch request (github issue #1)
- Updated RESTCONF reference
- Added note to open issues section to check github instead

B.3. 00 to 01

- Added text requiring support for Accept-Patch header, and removed 'Identification of YANG Patch capabilities' open issue.
- Removed 'location' leaf from yang-patch-status grouping
- Removed open issue 'Protocol independence' because the location leaf was removed.
- Removed open issue 'RESTCONF coupling' because there is no concern about a normative reference to RESTCONF. There may need to be a YANG 1.1 mechanism to allow protocol template usage (instead of grouping wrapper).
- Removed open issue 'Is the delete operation needed'. It was decided that both delete and remove should remain as operations and clients can choose which one to use. This is not an implementation burden on the server.
- Removed open issue 'global-errors needed'. It was decided that they are needed as defined because the global <ok/> is needed and the special key value for edit=global error only allows for 1 global error.

- Removed open issue 'Is location leaf needed'. It was decided that it is not needed so this leaf has been removed.

- Removed open issue 'Bulk editing support in yang-patch-status'. The 'location' leaf has been removed so this issue is no longer applicable.

- Removed open issue 'Edit list mechanism'. Added text to the 'edit' list description-stmt about how the individual edits must be processed. There is no concern about duplicate edits which cause intermediate results to be altered by subsequent edits in the same edit list.

B.4. bierman:yang-patch-00 to ietf:yang-patch-00

- Created open issues section

Appendix C. Open Issues

-- RFC Ed.: remove this section before publication.

Refer to the github issue tracker for any open issues:

https://github.com/netconf-wg/yang-patch/issues

Appendix D. Example YANG Module

The example YANG module used in this document represents a simple media jukebox interface. The "example-jukebox" YANG module is defined in [I-D.ietf-netconf-restconf].

YANG Tree Diagram for "example-jukebox" Module:

```
      +--rw jukebox?
         +--rw library
         |  +--rw artist [name]
         |  |  +--rw name     string
         |  |  +--rw album [name]
         |  |     +--rw name     string
         |  |     +--rw genre?   identityref
         |  |     +--rw year?    uint16
         |  |     +--rw admin
         |  |     |  +--rw label?              string
         |  |     |  +--rw catalogue-number?   string
         |  |     +--rw song [name]
         |  |        +--rw name        string
         |  |        +--rw location    string
         |  |        +--rw format?     string
         |  |        +--rw length?     uint32
         |  +--ro artist-count?   uint32
         |  +--ro album-count?    uint32
         |  +--ro song-count?     uint32
         +--rw playlist [name]
         |  +--rw name           string
         |  +--rw description?   string
         |  +--rw song [index]
         |     +--rw index    uint32
         |     +--rw id       instance-identifier
         +--rw player
            +--rw gap?   decimal64

      rpcs:

       +---x play
          +--ro input
             +--ro playlist       string
             +--ro song-number    uint32
```

D.1. YANG Patch Examples

This section includes RESTCONF examples. NETCONF examples are TBD. Most examples are shown in JSON encoding [RFC7158], and some are shown in XML encoding [W3C.REC-xml-20081126].

D.1.1. Add Resources: Error

The following example shows several songs being added to an existing album. Each edit contains one song. The first song already exists, so an error will be reported for that edit. The rest of the edits were not attempted, since the first edit failed.

Request from client:

```
PATCH /restconf/data/example-jukebox:jukebox/
   library/artist=Foo%20Fighters/album=Wasting%20Light HTTP/1.1
Host: example.com
Accept: application/yang.patch-status+json
Content-Type: application/yang.patch+json

{
  "ietf-yang-patch:yang-patch" : {
    "patch-id" : "add-songs-patch",
    "edit" : [
      {
        "edit-id" : 1,
        "operation" : "create",
        "target" : "/song",
        "value" : {
          "song" : {
            "name" : "Bridge Burning",
            "location" : "/media/bridge_burning.mp3",
            "format" : "MP3",
            "length" : 288
          }
        }
      },
      {
        "edit-id" : 2,
        "operation" : "create",
        "target" : "/song",
        "value" : {
          "song" : {
            "name" : "Rope",
            "location" : "/media/rope.mp3",
            "format" : "MP3",
            "length" : 259
          }
        }
      },
      {
        "edit-id" : 3,
        "operation" : "create",
        "target" : "/song",
        "value" : {
          "song" : {
            "name" : "Dear Rosemary",
            "location" : "/media/dear_rosemary.mp3",
            "format" : "MP3",
            "length" : 269
```

```
              }
            }
          }
        ]
      }
    }
```

Response from server:

```
  HTTP/1.1 409 Conflict
  Date: Mon, 23 Apr 2012 13:01:20 GMT
  Server: example-server
  Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
  Content-Type: application/yang.patch-status+json

  {
    "ietf-yang-patch:yang-patch-status" : {
      "patch-id" : "add-songs-patch",
      "edit-status" : {
        "edit" : [
          {
            "edit-id" : 1,
            "errors" : {
              "error" : [
                {
                  "error-type": "application",
                  "error-tag": "data-exists",
                  "error-path": "/example-jukebox:jukebox/library
                     /artist=Foo%20Fighters/album=Wasting%20Light
                     /song=Burning%20Light",
                  "error-message":
                    "Data already exists, cannot be created"
                }
              ]
            }
          }
        ]
      }
    }
```

D.1.2. Add Resources: Success

The following example shows several songs being added to an existing album.

- Each of 2 edits contains one song.

- Both edits succeed and new sub-resources are created

Request from client:

```
PATCH /restconf/data/example-jukebox:jukebox/
   library/artist=Foo%20Fighters/album=Wasting%20Light
   HTTP/1.1
Host: example.com
Accept: application/yang.patch-status+json
Content-Type: application/yang.patch+json

{
  "ietf-yang-patch:yang-patch" : {
    "patch-id" : "add-songs-patch-2",
    "edit" : [
      {
        "edit-id" : 1,
        "operation" : "create",
        "target" : "/song",
        "value" : {
          "song" : {
            "name" : "Rope",
            "location" : "/media/rope.mp3",
            "format" : "MP3",
            "length" : 259
          }
        }
      },
      {
        "edit-id" : 2,
        "operation" : "create",
        "target" : "/song",
        "value" : {
          "song" : {
            "name" : "Dear Rosemary",
            "location" : "/media/dear_rosemary.mp3",
            "format" : "MP3",
            "length" : 269
          }
        }
      }
    ]
  }
}
```

Response from server:

```
HTTP/1.1 200 Success
```

```
Date: Mon, 23 Apr 2012 13:01:20 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
Content-Type: application/yang.patch-status+json

{
  "ietf-yang-patch:yang-patch-status" : {
    "patch-id" : "add-songs-patch-2",
    "ok" : [null]
  }
}
```

D.1.3. Move list entry example

The following example shows a song being moved within an existing playlist. Song "1" in playlist "Foo-One" is being moved after song "3" in the playlist. The operation succeeds, so a non-error reply example can be shown.

Request from client:

```
PATCH /restconf/data/example-jukebox:jukebox/
  playlist=Foo-One   HTTP/1.1
Host: example.com
Accept: application/yang.patch-status+json
Content-Type: application/yang.patch+json

{
  "ietf-yang-patch:yang-patch" : {
    "patch-id" : "move-song-patch",
    "comment" : "Move song 1 after song 3",
    "edit" : [
      {
        "edit-id" : 1,
        "operation" : "move",
        "target" : "/song/1",
        "point" : "/song3",
        "where" : "after"
      }
    ]
  }
}
```

Response from server:

```
HTTP/1.1 400 OK
Date: Mon, 23 Apr 2012 13:01:20 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
Content-Type: application/yang.patch-status+json

{
  "ietf-restconf:yang-patch-status" : {
    "patch-id" : "move-song-patch",
    "ok" : [null]
  }
}
```

Authors' Addresses

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net