YANG Patch Media Type
draft-ietf-netconf-yang-patch-00

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 September 23, 2014.

Copyright Notice

Copyright (c) 2014 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 (http://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 BCP 78.
Table of Contents

1. Introduction .................................................. 3
   1.1. Terminology ............................................. 3
       1.1.1. NETCONF ........................................... 3
       1.1.2. HTTP ............................................... 4
       1.1.3. YANG ............................................... 4
       1.1.4. Terms .............................................. 5
       1.1.5. Tree Diagrams ...................................... 5

2. YANG Patch ..................................................... 6
   2.1. Target Resource .......................................... 6
   2.2. yang-patch Input ......................................... 6
   2.3. yang-patch-status Output ................................ 7
   2.4. Target Data Node ......................................... 7
   2.5. Edit Operations ......................................... 7
   2.6. Error Handling .......................................... 8

3. YANG Module .................................................... 9

4. IANA Considerations ........................................... 18
   4.1. YANG Module Registry .................................... 18
   4.2. application/yang.patch Media Types ..................... 18
   4.3. application/yang.patch-status Media Types ............... 18

5. Security Considerations ....................................... 20

6. Normative References ......................................... 21

Appendix A. Change Log ........................................... 22
   A.1. bierman:yang-patch-00 to ietf:yang-patch-00 ........... 22

Appendix B. Open Issues .......................................... 23
   B.1. Identification of YANG Patch capabilities .............. 23
   B.2. Protocol independence ................................... 23
   B.3. RESTCONF coupling ....................................... 23
   B.4. Delete operation needed? ................................ 24
   B.5. global-errors needed? .................................... 24
   B.6. Is location leaf needed? ................................ 24
   B.7. Bulk editing support in yang-patch-status ............. 24
   B.8. Edit list mechanism ...................................... 24

Appendix C. Example YANG Module ................................. 26
   C.1. YANG Patch Examples ..................................... 26
       C.1.1. Patch an Existing Data Resource .................. 27
       C.1.2. Add Resources: Error ............................... 27
       C.1.3. Add Resources: Success ............................. 29
       C.1.4. Move list entry example ............................ 31

Authors’ Addresses ............................................... 33
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 [RESTCONF] protocol 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
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. 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".

1.1.5. 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 'yang-patch' container 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.

For NETCONF, a YANG "rpc" statement must be defined. The "yang-patch" grouping MUST be included in the input parameters and the "yang-patch-status" grouping MUST be included in the output parameters.

2.1. Target Resource

The YANG Patch operation uses a conceptual root within a NETCONF configuration datastore to identify 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.

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
                       +--:(location)
                       |  +--rw location?   inet:uri
                       +--:(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.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>create a new data resource if it does not already exist or error</td>
</tr>
<tr>
<td>delete</td>
<td>delete a data resource if it already exists or error</td>
</tr>
<tr>
<td>insert</td>
<td>insert a new user-ordered data resource</td>
</tr>
<tr>
<td>merge</td>
<td>merge the edit value with the target data resource; create if it does not already exist</td>
</tr>
<tr>
<td>move</td>
<td>re-order the target data resource</td>
</tr>
<tr>
<td>replace</td>
<td>replace the target data resource with the edit value</td>
</tr>
<tr>
<td>remove</td>
<td>remove a data resource if it already exists or no error</td>
</tr>
</tbody>
</table>

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.
3. YANG Module

The "ietf-yang-patch" module defines conceptual definitions within groupings, which are not meant to be implemented as datastore contents by a server.

The "ietf-yang-types" and "ietf-inet_types" modules from [RFC6991] are used by this module for some type definitions.

The "ietf-restconf" module from [RESTCONF] is used by this module for a grouping definition.

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

<CODE BEGINS> file "ietf-yang-patch@2014-03-22.yang"

module ietf-yang-patch {
    prefix "ypatch";

    import ietf-inet-types { prefix inet; }
    import ietf-restconf { prefix rc; }

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

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

        WG Chair: Bert Wijnen
        <mailto:bertietf@bwijnen.net>

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

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

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

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

        Editor: Rex Fernando
        <mailto:rex@cisco.com>";

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) 2014 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-00.txt

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
revision 2014-03-22 {
    description
        "Initial revision.";
    reference
        "RFC XXXX: YANG Patch";
}

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.";
}
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.";

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.";
}
} // 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.";

      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.";
    }
    leaf location {
        type inet:uri;
        description
            "Contains the Location header value that would be returned if this edit causes a new resource to be created. If the edit identified by the same edit-id value was successfully invoked and a new resource was created, then this field will be returned instead of 'ok'.";
    }
    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.

    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
    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
5. Security Considerations

TBD
6. Normative References

draft-ietf-json-rfc4627bis-10 (work in progress),
December 2013.

[RESTCONF]   Bierman, A., Bjorklund, M., Watsen, K., and R. Fernando,
"RESTCONF Protocol", draft-ietf-netconf-restconf-00 (work
in progress), December 2013.

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

[RFC2616]   Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext

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

RFC 5789, March 2010.

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

and A. Bierman, Ed., "Network Configuration Protocol
(NETCONF)", RFC 6241, June 2011.

[RFC6991]   Schoenwaelder, J., "Common YANG Data Types", RFC 6991,
July 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. Change Log

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

A.1. bierman:yang-patch-00 to ietf:yang-patch-00

- Created open issues section
Appendix B. Open Issues

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

B.1. Identification of YANG Patch capabilities

How does a client know that the YANG Patch media type is supported?

Status: closed-update-pending

Resolution: the Accept-Patch header defined in RFC 5789 needs to be implemented by the server to advertise the media types supported for PATCH.

B.2. Protocol independence

- yang-patch/target and point leaves are type target-resource-offset. Is this syntax for a relative resource identifier acceptable for all protocols? Acceptable for just NETCONF and RESTCONF?

- yang-patch-status/location leaf (type inet:uri) Can this be ignored in a protocol such as NETCONF?

Status: open (pending location issue)

Solution proposal: Change the 'location' leaf to match the 'error-node' choice in RESTCONF:

    choice resource-identifier {
        leaf location-path {
            type instance-identifier;
        }
        leaf location-urlpath {
            type rc:data-resource-identifier;
        }
    }

B.3. RESTCONF coupling

- RESTCONF is a normative reference for YANG Patch, imported in YANG module for rc:errors grouping. Is this a problem for protocol independence?

Status: open
B.4. Delete operation needed?

o Is the delete operation needed?

Status: closed-update-pending

Resolution: The delete operation is not needed. The remove operation is good enough, and the ability to delete only if the target data node exists is not needed. The delete option is dependent on the treatment of default leafs and RESTCONF tries to hide the with-defaults details.

B.5. global-errors needed?

o There are 2 <errors> sections in yang-patch-status (global-errors/errors and edit-status/edit/errors). If a special ‘edit-id’ value was reserved, such as an empty string, it could be used in the ‘edit’ list to identify global errors

Status: open

B.6. Is location leaf needed?

The location leaf in each edit is used to identify the sub-resource created by the edit request. Is this leaf needed at all? The POST method needs to return a Location header but no such requirement exists for sub-resources created via PATCH.

Status: open

Solution proposal: remove location leaf from yang-patch-status.

B.7. Bulk editing support in yang-patch-status

What if the protocol is using a bulk edit (such as NETCONF-EX <edit2>) and the edit list is applied to multiple nodes? Should ‘leaf location’ be changed to ‘leaf-list location’?

Status: open (pending location issue)

Solution Proposal: the location leaf will be changed to a leaf-list in case multiple sub-resources are created in one edit operation.

B.8. Edit list mechanism

o Is the edit list approach better or worse than the ‘config’ subtree approach used in the <edit-config> operation? NETCONF does not have any constraints on how operation attributes are
nested, or whether duplicate nodes are allowed. There is no
specified processing order in NETCONF.

Status: open

- Are constraints needed (for either approach) in order to improve
  processing consistency and interoperability? E.g., no nested
  operations allowed, no duplicates allowed?

Status: open
Appendix C. 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 [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 song [name]
   |   |   |  |     |  ++--rw name     string
   |   |   |  |     |  ++--rw location string
   |   |   |  |     |  ++--rw format?  string
   |   |   |  |     |  ++--rw length?  uint32
   |   |   ++--ro artist-count?  uint32
   |   |   ++--ro album-count?  uint32
   |   |   ++--rw 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
```

C.1. YANG Patch Examples

This section includes RESTCONF examples. NETCONF examples are TBD. Most examples are shown in JSON encoding [JSON], and some are shown in XML encoding [W3C.REC-xml-20081126].
C.1.1. Patch an Existing Data Resource

To replace just the "year" field in the "album" resource (instead of replacing the entire resource), the client might send a plain patch as follows. Note that the request URI header line is wrapped for display purposes only:

```
PATCH /restconf/data/example-jukebox:jukebox/
    library/artist/Foo%20Fighters/album/Wasting%20Light HTTP/1.1
Host: example.com
If-Match: b8389233a4c
Content-Type: application/yang.data+json

{ "example-jukebox:year" : 2011 }
```

If the field is updated, the server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:49:30 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 17:49:30 GMT
ETag: b2788923da4c
```

The XML encoding for the same request might be:

```
PATCH /restconf/data/example-jukebox:jukebox/
    library/artist/Foo%20Fighters/album/Wasting%20Light HTTP/1.1
Host: example.com
If-Match: b8389233a4c
Content-Type: application/yang.data+xml

<year xmlns="http://example.com/ns/example-jukebox">2011</year>
```

C.1.2. 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-restconf: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-restconf: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"
                     }
                  ]
               }
            }
         }
      }
   }
}

C.1.3. 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/song/Burning%20Light HTTP/1.1
Host: example.com
Accept: application/yang.patch-status+json
Content-Type: application/yang.patch+json

{
   "ietf-restconf: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/1.1 409 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-restconf:yang-patch-status": {
      "patch-id": "add-songs-patch-2",
      "error": {
         "error-code": "conflict",
         "error-message": "PATCH was not accepted due to pre-existing object"
      }
   }
}
"ok": [null],
"edit-status": {
  "edit": [
    {
      "edit-id": 1,
      "location": "http://example.com/restconf/
                   data/example-jukebox:jukebox/library/artist/
                   Foo%20Fighters/album/Wasting%20Light/song/Rope"
    },
    {
      "edit-id": 2,
      "location": "http://example.com/restconf/
                   data/example-jukebox:jukebox/library/artist/
                   Foo%20Fighters/album/Wasting%20Light/song/
                   Dear%20Rosemary"
    }
  ]
}
}

C.1.4. 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-restconf: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],
        "edit-status" : {
            "edit" : [
                {
                    "edit-id" : 1,
                    "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

Rex Fernando
Cisco

Email: rex@cisco.com