Internet Engineering Task Force (IETF)
Request for Comments: 8072
Category: Standards Track
ISSN: 2070-1721

A. Bierman
YumaWorks
M. Bjorklund
Tail-f Systems
K. Watsen
Juniper Networks

February 2017

YANG Patch Media Type

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

There is a need for standard mechanisms to patch datastores defined in [RFC6241], which contain conceptual data that conforms to schema specified with YANG [RFC7950]. An "ordered 'edit' list" approach is needed to provide RESTCONF client developers with more precise RESTCONF client control of the edit procedure than the "plain patch" mechanism found in [RFC8040].

This document defines a media type for a YANG-based editing mechanism that can be used with the HTTP PATCH method [RFC5789]. YANG Patch is designed to support the RESTCONF protocol, defined in [RFC8040]. This document only specifies the use of the YANG Patch media type with the RESTCONF protocol.

It may be possible to use YANG Patch with other protocols besides RESTCONF. This is outside the scope of this document. For any protocol that supports the YANG Patch media type, if the entire patch document cannot be successfully applied, then the server MUST NOT apply any of the changes. It may be possible to use YANG Patch with datastore types other than a configuration datastore. This is outside the scope of this document.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.1.1. NETCONF

The following terms are defined in [RFC6241]:

- o configuration data
- o datastore
- o configuration datastore
- o protocol operation
- o running configuration datastore
- o state data
- o user

1.1.2. HTTP

The following terms are defined in [RFC7230]:

- header field
- message-body
- query
- request URI

The following terms are defined in [RFC7231]:

- method
- request
- resource

1.1.3. YANG

The following terms are defined in [RFC7950]:

- container
- data node
- leaf
- leaf-list
- list

1.1.4. RESTCONF

The following terms are defined in [RFC8040]:

- application/yang-data+xml
- application/yang-data+json
- data resource
- datastore resource
- patch

- RESTCONF capability
- target resource
- YANG data template

1.1.5. YANG Patch

The following terms are used within this document:

- RESTCONF client: a client that implements the RESTCONF protocol.
- RESTCONF server: a server that implements the RESTCONF protocol.
- YANG Patch: a conceptual edit request using the "yang-patch" YANG Patch template, defined in Section 3. In HTTP, refers to a PATCH method where a representation uses either the media type "application/yang-patch+xml" or "application/yang-patch+json".
- YANG Patch Status: a conceptual edit status response using the YANG "yang-patch-status" YANG data template, defined in Section 3. In HTTP, refers to a response message for a PATCH method, where it has a representation with either the media type "application/yang-data+xml" or "application/yang-data+json".
- YANG Patch template: similar to a YANG data template, except that it has a representation with the media type "application/yang-patch+xml" or "application/yang-patch+json".

1.1.6. Examples

Some protocol message lines within examples throughout this document are split into multiple lines for display purposes only. When a line ends with a backslash ("\") as the last character, the line is wrapped for display purposes. It is to be considered to be joined to the next line by deleting the backslash, the following line break, and the leading whitespace of the next line.

1.1.7. Tree Diagram Notations

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

- Brackets "[" and "]" enclose list keys.

- Abbreviations before data node names: "rw" means configuration data (read-write), "ro" means state data (read-only), and "x" means operation resource (executable).

- 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 RESTCONF server. The specific fields are defined in the YANG module in Section 3.

The YANG Patch operation is invoked by the RESTCONF client by sending a PATCH method request with a representation using either the media type "application/yang-patch+xml" or "application/yang-patch+json". This message-body representing the YANG Patch input parameters MUST be present.

YANG Patch has some features that are not possible with the "plain-patch" mechanism defined in RESTCONF [RFC8040]:

- YANG Patch allows multiple sub-resources to be edited within the same PATCH method.

- YANG Patch allows a more precise edit operation than the "plain patch" mechanism found in [RFC8040]. There are seven operations supported ("create", "delete", "insert", "merge", "move", "replace", and "remove").

- YANG Patch uses an "edit" list with an explicit processing order. The edits are processed in client-specified order, and error processing can be precise even when multiple errors occur in the same YANG Patch request.

The YANG Patch "patch-id" may be useful for debugging and SHOULD be present in any audit logging records generated by the RESTCONF server for a patch.

The RESTCONF server MUST return the "Accept-Patch" header field in an OPTIONS response, as specified in [RFC5789], which includes the media type for YANG Patch. This is needed by a client to determine the message-encoding formats supported by the server (e.g., XML, JSON, or both). The following is an example of an "Accept-Patch" header:

```
 Accept-Patch: application/yang-patch+xml,application/yang-patch+json
```

Note that YANG Patch can only edit data resources. The PATCH method cannot be used to replace the datastore resource. Although the "ietf-yang-patch" YANG module is written using YANG version 1.1 [RFC7950], an implementation of YANG Patch can be used with content defined in YANG version 1 [RFC6020] as well.

A YANG Patch can be encoded in XML format according to [W3C.REC-xml-20081126]. It can also be encoded in JSON according to "JSON Encoding of Data Modeled with YANG" [RFC7951]. If any metadata needs to be sent in a JSON message, it is encoded according to "Defining and Using Metadata with YANG" [RFC7952].

2.1. Target Resource

The YANG Patch operation uses the RESTCONF target resource URI to identify the resource that will be patched. This can be the datastore resource itself, i.e., "{+restconf}/data", to edit top-level configuration data resources, or it can be a configuration data resource within the datastore resource, e.g., "{+restconf}/data/ietf-interfaces:interfaces", to edit sub-resources within a top-level configuration data resource.

The target resource MUST identify exactly one resource instance. If more than one resource instance is identified, then the request MUST NOT be processed and a "400 Bad Request" error response MUST be sent by the server. If the target resource does not identify any existing resource instance, then the request MUST NOT be processed and a "404 Not Found" error response MUST be sent by the server.

Each edit with a YANG Patch identifies a target data node for the associated edit. This is described in Section 2.4.

2.2. yang-patch Request

A YANG Patch is identified by a unique "patch-id", and it may have an optional comment. A patch is an ordered collection of edits. Each edit is identified by an "edit-id", and it has an edit operation ("create", "delete", "insert", "merge", "move", "replace", or "remove") that is applied to the target resource. Each edit can be applied to a sub-resource "target" within the target resource. If the operation is "insert" or "move", then the "where" parameter indicates how the node is inserted or moved. For values "before" and "after", the "point" parameter specifies the data node insertion point.

The "merge", "replace", "create", "delete", and "remove" edit operations have exactly the same meanings as those defined for the "operation" attribute described in Section 7.2 of [RFC6241].

Each edit within a YANG Patch MUST identify exactly one data resource instance. If an edit represents more than one resource instance, then the request MUST NOT be processed and a "400 Bad Request" error response MUST be sent by the server. If the edit does not identify any existing resource instance and the operation for the edit is not "create", then the request MUST NOT be processed and a "404 Not Found" error response MUST be sent by the server. A "yang-patch-status" response MUST be sent by the server identifying the edit or edits that are not valid.

YANG Patch does not provide any access to specific datastores. How a server processes an edit if it is co-located with a Network Configuration Protocol (NETCONF) server that does provide access to individual datastores is left up to the implementation. A complete datastore cannot be replaced in the same manner as that provided by the <copy-config> operation defined in Section 7.3 of [RFC6241]. Only the specified nodes in a YANG Patch are affected.

A message-body representing the YANG Patch is sent by the RESTCONF 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:

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

2.3. yang-patch-status Response

A message-body representing the YANG Patch Status is returned to the RESTCONF 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; the syntax specification is defined in Section 3.

YANG tree diagram for "yang-patch-status" container:

```
     +---- yang-patch-status
           +---- patch-id       string
           +---- (global-status)?
           |  +--:(global-errors)
           |  |  +---- errors
           |  |     +---- error*
           |  |        +---- error-type       enumeration
           |  |        +---- error-tag        string
           |  |        +---- error-app-tag?   string
           |  |        +---- error-path?      instance-identifier
           |  |        +---- error-message?   string
           |  |        +---- error-info?
           |  +--:(ok)
           |     +---- ok?            empty
           +---- edit-status
              +---- edit* [edit-id]
                 +---- edit-id    string
                 +---- (edit-status-choice)?
                    +--:(ok)
                    |  +---- ok?        empty
                    +--:(errors)
                       +---- errors
                          +---- error*
                             +---- error-type       enumeration
                             +---- error-tag        string
                             +---- error-app-tag?   string
                             +---- error-path?      instance-identifier
                             +---- error-message?   string
                             +---- error-info?
```

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 treated as an absolute path expression identifying the target data node for the corresponding edit. The first node specified in the "target" leaf is a top-level data node defined within a YANG module. The "target" leaf MUST NOT contain a single forward slash ("/"), since this would identify the datastore resource, not a data resource.

If the target resource specified in the request URI identifies a configuration data resource, then the path string in the "target" leaf is treated as a relative path expression. The first node specified in the "target" leaf is a child configuration data node of the data node associated with the target resource. If the "target" leaf contains a single forward slash ("/"), then the target data node is the target resource data node.

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; if it already exists, return an error
delete	delete a data resource if it already exists; if it does not exist, return an 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	reorder the target data resource
replace	replace the target data resource with the edit value
remove	remove a data resource if it already exists

YANG Patch Edit Operations

2.6. Successful Edit Response Handling

If a YANG Patch is completed without errors, the RESTCONF server MUST return a "yang-patch-status" message with a "global-status" choice set to "ok".

Refer to Appendix A.1.2 for an example of a successful YANG Patch response.

2.7. Error Handling

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

If a YANG Patch is completed with errors, the RESTCONF server SHOULD return a "yang-patch-status" message. It is possible (e.g., within a distributed implementation) that an invalid request will be rejected before the YANG Patch edits are processed. In this case, the server MUST send the appropriate HTTP error response instead.

Refer to Appendix A.1.1 for an example of an error YANG Patch response.

2.8. ":yang-patch" RESTCONF Capability

A URI is defined to identify the YANG Patch extension to the base RESTCONF protocol. If the RESTCONF server supports the YANG Patch media type, then the ":yang-patch" RESTCONF capability defined in Section 4.3 MUST be present in the "capability" leaf-list in the "ietf-restconf-monitoring" module defined in [RFC8040].

3. YANG Module

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

The "ietf-restconf" module from [RFC8040] is used by this module for the "yang-data" extension definition.

```
<CODE BEGINS>

file "ietf-yang-patch@2017-02-22.yang"

module ietf-yang-patch {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-patch";
  prefix "ypatch";

  import ietf-restconf { prefix rc; }

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

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

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

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

     Author:   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) 2017 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 8072; see
           the RFC itself for full legal notices.";

      revision 2017-02-22 {
        description
          "Initial revision.";
        reference
          "RFC 8072: YANG Patch Media Type.";
      }

      typedef target-resource-offset {
        type string;
        description
          "Contains a data resource identifier string representing
           a sub-resource within the target resource.
           The document root for this expression is the
           target resource that is specified in the
           protocol operation (e.g., the URI for the PATCH request).

           This string is encoded according to the same rules as those
           for a data resource identifier in a RESTCONF request URI.";
        reference
           "RFC 8040, Section 3.5.3.";
      }

      rc:yang-data "yang-patch" {
        uses yang-patch;
      }

      rc:yang-data "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 YANG Patch operation.

            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 7950, Section 8.3.";

         leaf patch-id {
           type string;
           mandatory true;
           description
             "An arbitrary string provided by the client to identify
              the entire patch.  Error messages returned by the server
              that pertain to this patch will be identified by this
              'patch-id' value.  A client SHOULD attempt to generate
              unique 'patch-id' values to distinguish between
              transactions from multiple clients in any audit logs
              maintained by the server.";
         }

         leaf comment {
           type string;
           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 that pertain
                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.  The
                    'target' leaf identifies the data node to be
                    created, not the parent data node.";
               }
               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.  If the 'where'
                    parameter is set to 'before' or 'after', then
                    the 'point' parameter identifies the insertion
                    point for the target node.";
               }
               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.  If the 'where' parameter
                    is set to 'before' or 'after', then the 'point'
                    parameter identifies the insertion point to move
                    the target node.";
               }
               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 node for the edit
                operation.  If the target has the value '/', then
                the target data node is the target resource.
                The target node MUST identify a data resource,
                not the datastore resource.";
           }

           leaf point {
             when "(../operation = 'insert' or ../operation = 'move')"
                + "and (../where = 'before' or ../where = 'after')" {
               description
                 "This 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
                 "This 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'.";
           }

           anydata value {
             when "../operation = 'create' "
                + "or ../operation = 'merge' "
                + "or ../operation = 'replace' "
                + "or ../operation = 'insert'" {
               description
                 "The anydata 'value' is only used for 'create',
                  'merge', 'replace', and 'insert' operations.";
             }
             description
               "Value used for this edit operation.  The anydata 'value'
                contains 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 contains 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 a 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;
           mandatory true;
           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 that
                are 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. Registrations for New URI and YANG Module

This document registers one URI as a namespace in the "IETF XML Registry" [RFC3688]. It follows the format in RFC 3688.

```
      URI: urn:ietf:params:xml:ns:yang:ietf-yang-patch
      Registrant Contact: The IESG.
      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
      reference:    RFC 8072
```

4.2. Media Types

4.2.1. Media Type "application/yang-patch+xml"

Type name: application

Subtype name: yang-patch+xml

Required parameters: None

Optional parameters: None

Encoding considerations: 8-bit
The "utf-8" charset is always used for this type.
Each conceptual YANG data node is encoded according to the XML Encoding Rules and Canonical Format for the specific YANG data node type defined in [RFC7950].
In addition, the "yang-patch" YANG Patch template found in RFC 8072 defines the structure of a YANG Patch request.

Security considerations: Security considerations related to the generation and consumption of RESTCONF messages are discussed in Section 5 of RFC 8072.
Additional security considerations are specific to the semantics of particular YANG data models. Each YANG module is expected to specify security considerations for the YANG data defined in that module.

Interoperability considerations: RFC 8072 specifies the format of conforming messages and the interpretation thereof.

Published specification: RFC 8072

Applications that use this media type: Instance document data parsers used within a protocol or automation tool that utilize the YANG Patch data structure.

Fragment identifier considerations: The syntax and semantics of fragment identifiers are the same as the syntax and semantics specified for the "application/xml" media type.

Additional information:

Deprecated alias names for this type: N/A
Magic number(s): N/A
File extension(s): None
Macintosh file type code(s): "TEXT"

Person & email address to contact for further information: See the Authors' Addresses section of RFC 8072.

Intended usage: COMMON

Restrictions on usage: N/A

Author: See the Authors' Addresses section of RFC 8072.

Change controller: Internet Engineering Task Force (mailto:iesg@ietf.org).

Provisional registration? (standards tree only): no

4.2.2. Media Type "application/yang-patch+json"

Type name: application

Subtype name: yang-patch+json

Required parameters: None

Optional parameters: None

Encoding considerations: 8-bit
The "utf-8" charset is always used for this type.
Each conceptual YANG data node is encoded according to RFC 7951. A metadata annotation is encoded according to RFC 7952. In addition, the "yang-patch" YANG Patch template found in RFC 8072 defines the structure of a YANG Patch request.

Security considerations: Security considerations related to the generation and consumption of RESTCONF messages are discussed in Section 5 of RFC 8072.
Additional security considerations are specific to the semantics of particular YANG data models. Each YANG module is expected to specify security considerations for the YANG data defined in that module.

Interoperability considerations: RFC 8072 specifies the format of conforming messages and the interpretation thereof.

Published specification: RFC 8072

Applications that use this media type: Instance document data parsers used within a protocol or automation tool that utilize the YANG Patch data structure.

Fragment identifier considerations: The syntax and semantics of fragment identifiers are the same as the syntax and semantics specified for the "application/json" media type.

Additional information:

Deprecated alias names for this type: N/A
Magic number(s): N/A
File extension(s): None
Macintosh file type code(s): "TEXT"

Person & email address to contact for further information: See the Authors' Addresses section of RFC 8072.

Intended usage: COMMON

Restrictions on usage: N/A

Author: See the Authors' Addresses section of RFC 8072.

Change controller: Internet Engineering Task Force (mailto:iesg@ietf.org).

Provisional registration? (standards tree only): no

4.3. RESTCONF Capability URNs

This document registers one capability identifier in the "RESTCONF Capability URNs" registry [RFC8040]. The review policy for this registry is "IETF Review" [RFC5226].

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 [RFC8040]. This document defines edit processing instructions for a variant of the PATCH method, as used within the RESTCONF protocol. Message integrity is provided by the RESTCONF protocol. There is no additional capability to validate that a patch has not been altered.

It may be possible to use YANG Patch with other protocols besides RESTCONF; this topic is outside the scope of this document.

For RESTCONF, both the client and server MUST be authenticated according to Section 2 of [RFC8040]. It is important for RESTCONF 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 PATCH request MUST be applied atomically, as specified in Section 2 of [RFC5789].

A RESTCONF server implementation SHOULD attempt to prevent system disruption due to incremental processing of the YANG Patch "edit" list. It may be possible to construct an attack on such a RESTCONF server, which relies on the edit processing order mandated by YANG Patch. A server SHOULD apply only the fully validated configuration to the underlying system. For example, an "edit" list that deleted an interface and then recreated it could cause system disruption if the "edit" list was incrementally applied.

A RESTCONF server implementation SHOULD attempt to prevent system disruption due to excessive resource consumption required to fulfill YANG Patch edit requests. On such an implementation, it may be possible to construct an attack that attempts to consume all available memory or other resource types.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <http://www.rfc-editor.org/info/rfc5789>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7230] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <http://www.rfc-editor.org/info/rfc7950>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <http://www.rfc-editor.org/info/rfc7951>.

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

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <http://www.rfc-editor.org/info/rfc8040>.

[W3C.REC-xml-20081126]
Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "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>.

6.2. Informative References

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

Appendix A. 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 [RFC8040].

YANG tree diagram for the "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
```

A.1. YANG Patch Examples

This section includes RESTCONF examples. Most examples are shown in JSON encoding [RFC7159], and some are shown in XML encoding [W3C.REC-xml-20081126].

A.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. XML encoding is used in this example.

Request from the RESTCONF client:

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

      <yang-patch xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-patch">
        <patch-id>add-songs-patch</patch-id>
        <edit>
          <edit-id>edit1</edit-id>
          <operation>create</operation>
          <target>/song=Bridge%20Burning</target>
          <value>
            <song xmlns="http://example.com/ns/example-jukebox">
              <name>Bridge Burning</name>
              <location>/media/bridge_burning.mp3</location>
              <format>MP3</format>
              <length>288</length>
            </song>
          </value>
        </edit>
        <edit>
          <edit-id>edit2</edit-id>
          <operation>create</operation>
          <target>/song=Rope</target>
          <value>
            <song xmlns="http://example.com/ns/example-jukebox">
              <name>Rope</name>
              <location>/media/rope.mp3</location>
              <format>MP3</format>
              <length>259</length>
            </song>
          </value>
        </edit>
        <edit>
          <edit-id>edit3</edit-id>
          <operation>create</operation>
          <target>/song=Dear%20Rosemary</target>
          <value>
            <song xmlns="http://example.com/ns/example-jukebox">
              <name>Dear Rosemary</name>
              <location>/media/dear_rosemary.mp3</location>
              <format>MP3</format>
              <length>269</length>
            </song>
          </value>
        </edit>
      </yang-patch>
```

   XML response from the RESTCONF server:

```
      HTTP/1.1 409 Conflict
      Date: Thu, 26 Jan 2017 20:56:30 GMT
      Server: example-server
      Last-Modified: Thu, 26 Jan 2017 20:56:30 GMT
      Content-Type: application/yang-data+xml

      <yang-patch-status
         xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-patch">
        <patch-id>add-songs-patch</patch-id>
        <edit-status>
          <edit>
             <edit-id>edit1</edit-id>
             <errors>
                <error>
                   <error-type>application</error-type>
                   <error-tag>data-exists</error-tag>
                   <error-path
                     xmlns:jb="http://example.com/ns/example-jukebox">
                     /jb:jukebox/jb:library
                     /jb:artist[jb:name='Foo Fighters']
                     /jb:album[jb:name='Wasting Light']
                     /jb:song[jb:name='Bridge Burning']
                   </error-path>
                   <error-message>
                     Data already exists; cannot be created
                   </error-message>
                </error>
             </errors>
          </edit>
        </edit-status>
      </yang-patch-status>
```

JSON response from the RESTCONF server:

The following response is shown in JSON format to highlight the difference in the "error-path" object encoding. For JSON, the instance-identifier encoding specified in [RFC7951] is used.

```
   HTTP/1.1 409 Conflict
   Date: Thu, 26 Jan 2017 20:56:30 GMT
   Server: example-server
   Last-Modified: Thu, 26 Jan 2017 20:56:30 GMT
   Content-Type: application/yang-data+json

   {
     "ietf-yang-patch:yang-patch-status" : {
       "patch-id" : "add-songs-patch",
       "edit-status" : {
         "edit" : [
           {
             "edit-id" : "edit1",
             "errors" : {
               "error" : [
                 {
                   "error-type": "application",
                   "error-tag": "data-exists",
                   "error-path": "/example-jukebox:jukebox/library\
                      /artist[name='Foo Fighters']\
                      /album[name='Wasting Light']\
                      /song[name='Bridge Burning']",
                   "error-message":
                     "Data already exists; cannot be created"
                 }
               ]
             }
           }
         ]
       }
     }
   }
```

A.1.2. Add Resources: Success

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

- o Each of two edits contains one song.

- o Both edits succeed, and new sub-resources are created.

Request from the RESTCONF client:

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

      {
        "ietf-yang-patch:yang-patch" : {
          "patch-id" : "add-songs-patch-2",
          "edit" : [
            {
              "edit-id" : "edit1",
              "operation" : "create",
              "target" : "/song=Rope",
              "value" : {
                "song" : [
                  {
                    "name" : "Rope",
                    "location" : "/media/rope.mp3",
                    "format" : "MP3",
                    "length" : 259
                  }
                ]
              }
            },
            {
              "edit-id" : "edit2",
              "operation" : "create",
              "target" : "/song=Dear%20Rosemary",
              "value" : {
                "song" : [
                  {
                    "name" : "Dear Rosemary",
                    "location" : "/media/dear_rosemary.mp3",
                    "format" : "MP3",
                    "length" : 269
                  }
                ]
              }
            }
          ]
        }
      }
```

Response from the RESTCONF server:

```
      HTTP/1.1 200 OK
      Date: Thu, 26 Jan 2017 20:56:30 GMT
      Server: example-server
      Last-Modified: Thu, 26 Jan 2017 20:56:30 GMT
      Content-Type: application/yang-data+json

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

A.1.3. Insert List Entry

The following example shows a song being inserted within an existing playlist. Song "6" in playlist "Foo-One" is being inserted after song "5" in the playlist. The operation succeeds, so a non-error reply can be provided.

Request from the RESTCONF client:

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

      {
        "ietf-yang-patch:yang-patch" : {
          "patch-id" : "insert-song-patch",
          "comment" : "Insert song 6 after song 5",
          "edit" : [
            {
              "edit-id" : "edit1",
              "operation" : "insert",
              "target" : "/song=6",
              "point" : "/song=5",
              "where" : "after",
              "value" : {
                "example-jukebox:song" : [
                  {
                    "index" : 6,
                    "id" : "/example-jukebox:jukebox/library\
                      /artist[name='Foo Fighters']\
                      /album[name='Wasting Light']\
                      /song[name='Bridge Burning']"
                  }
                ]
              }
            }
          ]
        }
```

Response from the RESTCONF server:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Last-Modified: Thu, 26 Jan 2017 20:56:30 GMT
Content-Type: application/yang-data+json

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

A.1.4. Move List Entry

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. Note that no "value" parameter is needed for a "move" operation. The operation succeeds, so a non-error reply can be provided.

Request from the RESTCONF client:

```
PATCH /restconf/data/example-jukebox:jukebox/\
  playlist=Foo-One HTTP/1.1
Host: example.com
Accept: application/yang-data+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" : "edit1",
        "operation" : "move",
        "target" : "/song=1",
        "point" : "/song=3",
        "where" : "after"
      }
    ]
  }
}
```

Response from the RESTCONF server:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Last-Modified: Thu, 26 Jan 2017 20:56:30 GMT
Content-Type: application/yang-data+json

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

A.1.5. Edit Datastore Resource

The following example shows how three top-level data nodes from different modules can be edited at the same time.

Example module "foo" defines leaf X. Example module "bar" defines container Y, with child leafs A and B. Example module "baz" defines list Z, with key C and child leafs D and E.

Request from the RESTCONF client:

```
PATCH /restconf/data HTTP/1.1
Host: example.com
Accept: application/yang-data+json
Content-Type: application/yang-patch+json

{
  "ietf-yang-patch:yang-patch" : {
    "patch-id" : "datastore-patch-1",
    "comment" : "Edit 3 top-level data nodes at once",
    "edit" : [
      {
        "edit-id" : "edit1",
        "operation" : "create",
        "target" : "/foo:X",
        "value" : {
          "foo:X" : 42
        }
      },
      {
        "edit-id" : "edit2",
        "operation" : "merge",
        "target" : "/bar:Y",
        "value" : {
          "bar:Y" : {
            "A" : "test1",
            "B" : 99
          }
        }
      },
      {
        "edit-id" : "edit3",
        "operation" : "replace",
        "target" : "/baz:Z=2",
        "value" : {
          "baz:Z" : [
            {
              "C" : 2,
              "D" : 100,
              "E" : false
            }
          ]
        }
      }
    ]
  }
}
```

Response from the RESTCONF server:

```
      HTTP/1.1 200 OK
      Date: Thu, 26 Jan 2017 20:56:30 GMT
      Server: example-server
      Last-Modified: Thu, 26 Jan 2017 20:55:30 GMT
      Content-Type: application/yang-data+json

      {
        "ietf-yang-patch:yang-patch-status" : {
          "patch-id" : "datastore-patch-1",
          "ok" : [null]
        }
      }
```

Acknowledgements

The authors would like to thank Rex Fernando for his contributions to this document.

Contributions to this material by Andy Bierman are based upon work supported by the United States Army, Space & Terrestrial Communications Directorate (S&TCD) under Contract No. W15P7T-13-C-A616. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the S&TCD.

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