Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 17, 2014

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

February 13, 2014

RESTCONF Protocol
draft-bierman-netconf-restconf-04

Abstract

This document describes a REST-like protocol that provides a programmatic interface over HTTP for accessing data defined in YANG, using the datastores defined in NETCONF.

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 17, 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
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction 5
1.1. Simple Subset of NETCONF Functionality 5
1.2. Data Model Driven API 6
1.3. Terminology 8
1.3.1. NETCONF 8
1.3.2. HTTP 8
1.3.3. YANG 9
1.3.4. Terms 9
1.3.5. Tree Diagrams 11
2. Operations 12
2.1. OPTIONS 12
2.2. HEAD 13
2.3. GET 13
2.4. POST 14
2.4.1. Create Resource Mode 14
2.4.2. Invoke Operation Mode 15
2.5. PUT 16
2.6. PATCH 17
2.7. DELETE 19
2.8. Query Parameters 19
3. Messages 21
3.1. Request URI Structure 21
3.2. Message Headers 22
3.3. Message Encoding 23
3.4. RESTCONF Meta-Data 23
3.4.1. JSON Encoding of RESTCONF Meta-Data 24
3.5. Return Status 26
3.6. Message Caching 26
4. Resources 27
4.1. RESTCONF Resource Types 27
4.2. Resource Discovery 28
4.3. API Resource (/restconf) 28
4.3.1. /restconf/data 29
4.3.2. /restconf/modules 30
4.3.3. /restconf/operations 31
4.3.4. /restconf/streams 31
4.3.5. /restconf/version 31
4.4. Datastore Resource 31
4.4.1. Edit Collision Detection 32
4.5. Data Resource 33
4.5.1. Encoding YANG Instance Identifiers in the Request

URI 33
4.5.2. Defaults Handling 36
4.6. Operation Resource 36
4.6.1. Encoding Operation Input Parameters 37
4.6.2. Encoding Operation Output Parameters 38
4.7. Schema Resource 39
4.8. Stream Resource 39
5. Notifications 41
5.1. Server Support 41
5.2. Event Streams 41
5.3. Subscribing to Receive Notifications 42
5.3.1. NETCONF Event Stream 43
5.4. Receiving Event Notifications 43
6. Error Reporting 45
6.1. Error Response Message 46
7. RESTCONF module 49
8. IANA Considerations 63
8.1. YANG Module Registry 63
8.2. application/yang Media Sub Types 63
9. Security Considerations 65
10. References 66
10.1. Normative References 66
10.2. Informative References 67
Appendix A. Change Log 68
A.1. 03 to 04 68
A.2. 02 to 03 68
A.3. 01 to 02 69
A.4. 00 to 01 69
A.5. YANG-API-01 to RESTCONF-00 70
Appendix B. Open Issues 72
B.1. message-id 72
B.2. select parameter 72
B.3. server support verification 72
B.4. error media type 72
B.5. additional datastores 72
B.6. PATCH media type discovery 72
B.7. RESTCONF version 72
B.8. YANG to resource mapping 73
B.9. .well-known usage 73
B.10. _self links for HATEOAS support 74
B.11. netconf-state monitoring support 74
B.12. secure transport 74
Appendix C. Example YANG Module 75
C.1. example-jukebox YANG Module 75
Appendix D. RESTCONF Message Examples 81
D.1. Resource Retrieval Examples 81
D.1.1. Retrieve the Top-level API Resource 81
D.1.2. Retrieve The Server Module Information 83

D.2. Edit Resource Examples 85
D.2.1. Create New Data Resources 85
D.2.2. Detect Resource Entity Tag Change 86
D.3. Query String Parameter Examples 86
D.3.1. "content" Parameter 86
D.3.2. "depth" Parameter 89
D.3.3. "filter" Parameter 92
D.3.4. "insert" Parameter 93
D.3.5. "point" Parameter 94
D.3.6. "select" Parameter 94
D.3.7. "start-time" Parameter 95
D.3.8. "stop-time" Parameter 95
Authors' Addresses 96

1. Introduction

There is a need for standard mechanisms to allow WEB applications to access the configuration data, operational data, data-model specific protocol operations, and notification events within a networking device, in a modular and extensible manner.

This document describes a REST-like protocol called RESTCONF, running over HTTP [RFC2616], for accessing data defined in YANG [RFC6020], using datastores defined in NETCONF [RFC6241].

The NETCONF protocol defines configuration datastores and a set of Create, Retrieve, Update, Delete (CRUD) operations that can be used to access these datastores. The YANG language defines the syntax and semantics of datastore content, operational data, protocol operations, and notification events. REST-like operations are used to access the hierarchical data within a datastore.

A REST-like API can be created that provides CRUD operations on a NETCONF datastore containing YANG-defined data. This can be done in a simplified manner, compatible with HTTP and REST-like design principles. Since NETCONF protocol operations are not relevant, the user should not need any prior knowledge of NETCONF in order to use the REST-like API.

Configuration data and state data are exposed as resources that can be retrieved with the GET method. Resources representing configuration data can be modified with the DELETE, PATCH, POST, and PUT methods. Data is encoded with either XML [W3C.REC-xml-20081126] or JSON [JSON].

Data-model specific protocol operations defined with the YANG "rpc" statement can be invoked with the POST method. Data-model specific notification events defined with the YANG "notification" statement can be accessed.

1.1. Simple Subset of NETCONF Functionality

The framework and meta-model used for a REST-like API does not need to mirror those used by the NETCONF protocol, but it needs to be compatible with NETCONF. A simplified framework and protocol is needed that utilizes the three NETCONF datastores (candidate, running, startup), but hides the complexity of multiple datastores from the client.

A simplified transaction model is needed that allows basic CRUD operations on a hierarchy of conceptual resources. This represents a limited subset of the transaction capabilities of the NETCONF

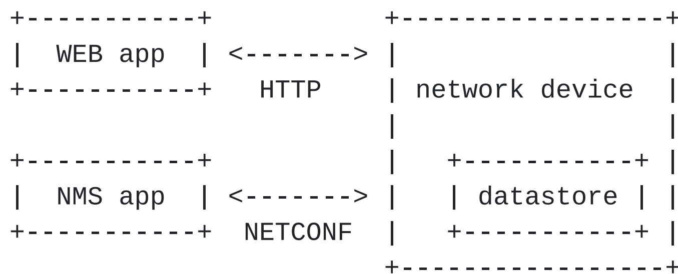
protocol.

Applications that require more complex transaction capabilities might consider NETCONF instead of RESTCONF. The following transaction features are not directly provided in RESTCONF:

- o datastore locking (full or partial)
- o candidate datastore
- o startup datastore
- o validate operation
- o confirmed-commit procedure

The REST-like API is not intended to replace NETCONF, but rather provide an additional simplified interface that follows REST-like principles and is compatible with a resource-oriented device abstraction.

The following figure shows the system components:



```
+-----------+           +-----------------+
|  WEB app  | <-------> |                 |
+-----------+   HTTP    | network device  |
                        |                 |
+-----------+           |   +-----------+ |
|  NMS app  | <-------> |   | datastore | |
+-----------+  NETCONF  |   +-----------+ |
                        +-----------------+
```

1.2. Data Model Driven API

RESTCONF combines the simplicity of a REST-like API over HTTP with the predictability and automation potential of a schema-driven API.

A REST-like client using HATEOAS principles would not use any data modeling language to define the application-specific content of the API. The client would discover each new child resource as it traverses the URIs returned as Location IDs to discover the server capabilities.

This approach has 3 significant weaknesses with regards to control of complex networking devices:

- o inefficient performance: configuration APIs will be quite complex and may require thousands of protocol messages to discover all the schema information. Typically the data type information has to be passed in the protocol messages, which is also wasteful overhead.

- o no data model richness: without a data model, the schema-level semantics and validation constraints are not available to the application.

- o no tool automation: API automation tools need some sort of content schema to function. Such tools can automate various programming and documentation tasks related to specific data models.

Data model modules such as YANG modules serve as an "API contract" that will be honored by the server. An application designer can code to the data model, knowing in advance important details about the exact protocol operations and datastore content a conforming server implementation will support.

RESTCONF provides the YANG module capability information supported by the server, in case the client wants to use it. The URIs for custom protocol operations and datastore content are predictable, based on the YANG module definitions.

Operational experience with CLI and SNMP indicates that operators learn the 'location' of specific service or device related data and do not expect such information to be arbitrary and discovered each time the client opens a management session to a server.

The RESTCONF protocol operates on a conceptual datastore defined with the YANG data modeling language. The server lists each YANG module it supports under "/restconf/modules" in the top-level API resource type, using a structure based on the YANG module capability URI format defined in [RFC6020].

The conceptual datastore contents, data-model-specific operations and notification events are identified by this set of YANG module resources. All RESTCONF content identified as either a data resource, operation resource, or event stream resource is defined with the YANG language.

The classification of data as configuration or non-configuration is derived from the YANG "config" statement. Data ordering behavior is derived from the YANG "ordered-by" statement.

The RESTCONF datastore editing model is simple and direct, similar to the behavior of the ":writable-running" capability in NETCONF. Each RESTCONF edit of a datastore resource is activated upon successful

completion of the transaction.

1.3. 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].

1.3.1. 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

1.3.2. HTTP

The following terms are defined in [RFC2616]:

- entity tag
- fragment
- header line
- message body

o method

o path

o query

o request URI

o response body

1.3.3. YANG

The following terms are defined in [RFC6020]:

o container

o data node

o key leaf

o leaf

o leaf-list

o list

o presence container (or P-container)

o RPC operation (now called protocol operation)

o non-presence container (or NP-container)

o ordered-by system

o ordered-by user

1.3.4. Terms

The following terms are used within this document:

o API resource: a resource with the media type "application/yang.api+xml" or "application/yang.api+json". API resources can only be edited by the server.

o data resource: a resource with the media type "application/yang.data+xml" or "application/yang.data+json". Data resources can be edited by clients or the server. All YANG data node types can be data resources. YANG terminal nodes cannot contain sub-

resources.

- datastore resource: a resource with the media type "application/yang.datastore+xml" or "application/yang.datastore+json". Represents a configuration datastore.

- edit operation: a RESTCONF operation on a data resource using the POST, PUT, PATCH, or DELETE method.

- event stream resource: a resource with the media type "application/yang.stream+xml" or "application/yang.stream+json". This resource represents an SSE (Server-Sent Events) event stream. The content consists of text using the media type "text/event-stream", as defined by the HTML5 specification. Each event represents one <notification> message generated by the server. It contains a conceptual system or data-model specific event that is delivered within a notification event stream.

- operation: the conceptual RESTCONF operation for a message, derived from the HTTP method, request URI, headers, and message body.

- operation resource: a resource with the media type "application/yang.operation+xml" or "application/yang.operation+json".

- patch: a generic PATCH method on the target datastore or data resource. The media type of the message body content will identify the patch type in use.

- plain patch: a PATCH method where the media type is "application/yang.data+xml" or "application/yang.data+json".

- query parameter: a parameter (and its value if any), encoded within the query component of the request URI.

- requested data nodes: the set of data resources identified by the target resource, or the "select" query parameter if it is present.

- resource: a conceptual object representing a manageable component within a device. Refers to the resource itself of the resource and all its sub-resources.

- retrieval request: a request using the GET or HEAD methods.

- target resource: the resource that is associated with a particular message, identified by the "path" component of the request URI.

- o unified datastore: A conceptual representation of the device running configuration. The server will hide all NETCONF datastore details for edit operations, such as the ":candidate" and ":startup" capabilities.

- o YANG schema resource: a resource with the media type "application/yang". The YANG representation of the schema can be retrieved by the client with the GET method.

- o YANG terminal node: a YANG node representing a leaf, leaf-list, or anyxml definition.

1.3.5. 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:

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

- o Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).

- o Symbols after data node names: "?" means an optional node and "*" denotes a "list" and "leaf-list".

- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

- o Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Operations

The RESTCONF protocol uses HTTP methods to identify the CRUD operation requested for a particular resource. The following table shows how the RESTCONF operations relate to NETCONF protocol operations:

RESTCONF	NETCONF
OPTIONS	none
HEAD	none
GET	<get-config>, <get>
POST	<edit-config> (operation="create")
PUT	<edit-config> (operation="replace")
PATCH	<edit-config> (operation="merge")
DELETE	<edit-config> (operation="delete")

Table 1: CRUD Methods in RESTCONF

The NETCONF "remove" operation attribute is not supported by the HTTP DELETE method. The resource must exist or the DELETE method will fail. The PATCH method is equivalent to a "merge" operation for a plain PATCH method.

Access control mechanims may be used to limit what operations can be used. In particular, RESTCONF is compatible with the NETCONF Access Control Model (NACM) [RFC6536], as there is a specific mapping between RESTCONF and NETCONF operations, defined in Table 1. The resource path needs to be converted internally by the server to the corresponding

YANG instance-identifier. Using this information, the server can apply the NACM access control rules to RESTCONF messages.

The server MUST NOT allow any operation to any resources that the client is not authorized to access.

Implementation of all methods (except PATCH) are defined in [RFC2616]. This section defines the RESTCONF protocol usage for each HTTP method.

2.1. OPTIONS

The OPTIONS method is sent by the client to discover which methods are supported by the server for a specific resource. If supported,

it SHOULD be implemented for all media types. The server SHOULD implement this method, however the same information could be extracted from the YANG modules and the RESTCONF protocol specification.

2.2. HEAD

The HEAD method is sent by the client to retrieve just the headers that would be returned for the comparable GET method, without the response body. It is supported for all resource types, except operation resources.

The request MUST contain a request URI that contains at least the entry point component. The same query parameters supported by the GET method are supported by the HEAD method.

The access control behavior is enforced as if the method was GET instead of HEAD. The server MUST respond the same as if the method was GET instead of HEAD, except that no response body is included.

2.3. GET

The GET method is sent by the client to retrieve data and meta-data for a resource. It is supported for all resource types, except operation resources. The request MUST contain a request URI that contains at least the entry point component.

The server MUST NOT return any data resources for which the user does not have read privileges. If the user is not authorized to read any portion of the target resource, an error response containing a "403 Forbidden" Status-Line is returned to the client.

If the user is authorized to read some but not all of the target resource, the unauthorized content is omitted from the response message body, and the authorized content is returned to the client.

Example:

The client might request the response headers for a JSON representation of the "library" resource:

```
GET /restconf/data/example-jukebox:jukebox/
  library/artist/Foo%20Fighters/album  HTTP/1.1
Host: example.com
Accept: application/yang.data+json
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+json
Cache-Control: no-cache
Pragma: no-cache
ETag: a74eefc993a2b
Last-Modified: Mon, 23 Apr 2012 11:02:14 GMT

{
  "album" : {
    "name" : "Wasting Light",
    "genre" : "example-jukebox:alternative",
    "year" : 2011
  }
}
```

Refer to @ex-create@ for more resource creation examples.

2.4. POST

The POST method is sent by the client to create a data resource or invoke an operation resource. The server uses the target resource media type to determine how to process the request.

Type	Description
Datastore	Create a top-level configuration data resource
Data	Create a configuration data sub-resource
Operation	Invoke a protocol operation

Resource Types that Support POST

The request MUST contain a request URI that contains a target resource which identifies a datastore, data, or operation resource type.

2.4.1. Create Resource Mode

If the target resource type is a datastore or data resource, then the POST is treated as a request to create a resource or sub-resource. The message body is expected to contain the content of a child resource to create within the parent (target resource).

The "insert" and "point" query parameters are supported by the POST method for datastore and data resource types, as specified in the

YANG definition in Section 7.

If the POST method succeeds, a "201 Created" Status-Line is returned and there is no response message body. A "Location" header identifying the child resource that was created MUST be present in the response in this case.

If the user is not authorized to create the target resource, an error response containing a "403 Forbidden" Status-Line is returned to the client. All other error responses are handled according to the procedures defined in Section 6.

Example:

To create a new "jukebox" resource, the client might send:

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

{ "example-jukebox:jukebox" : [null] }
```

If the resource is created, the server might respond as follows:

```
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Location: http://example.com/restconf/data/example-jukebox:jukebox
Last-Modified: Mon, 23 Apr 2012 17:01:00 GMT
ETag: b3a3e673be2
```

2.4.2. Invoke Operation Mode

If the target resource type is an operation resource, then the POST method is treated as a request to invoke that operation. The message body (if any) is processed as the operation input parameters. Refer to Section 4.6 for details on operation resources.

If the POST method succeeds, a "200 OK" Status-Line is returned if there is a response message body, and a "204 No Content" Status-Line is returned if there is no response message body.

If the user is not authorized to invoke the target operation, an error response containing a "403 Forbidden" Status-Line is returned to the client. All other error responses are handled according to the procedures defined in Section 6.

Example:

In this example, the client is invoking the "play" operation defined in the "example-jukebox" YANG module.

A client might send a "play" request as follows:

```
POST /restconf/operations/example-jukebox:play   HTTP/1.1
Host: example.com
Content-Type: application/yang.operation+json

{
  "example-jukebox:input" : {
    "playlist" : "Foo-One",
    "song-number" : 2
  }
}
```

The server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:50:00 GMT
Server: example-server
```

2.5. PUT

The PUT method is sent by the client to create or replace the target resource.

The request MUST contain a request URI that contains a target resource that identifies the data resource to create or replace.

If the resource instance does not exist, and it represents a valid instance the server could create with a POST request, then the server SHOULD create it.

The message body is expected to contain the content used to create or replace the target resource.

The "insert" and "point" query parameters are supported by the PUT method for data resources, as specified in the YANG definition in Section 7.

Consistent with [RFC2616], if the PUT method creates a new resource, a "201 Created" Status-Line is returned. If an existing resource is modified, either "200 OK" or "204 No Content" are returned.

If the user is not authorized to create or replace the target resource an error response containing a "403 Forbidden" Status-Line is returned to the client. All other error responses are handled

according to the procedures defined in Section 6.

Example:

An "album" sub-resource defined in the "example-jukebox" YANG module is replaced or created if it does not already exist.

To replace the "album" resource contents, the client might send as follows. Note that the request URI header line is wrapped for display purposes only:

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

{
  "example-jukebox:album" : {
    "name" : "Wasting Light",
    "genre" : "example-jukebox:alternative",
    "year" : 2011
  }
}
```

If the resource is updated, the server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:04:00 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 17:04:00 GMT
ETag: b27480aeda4c
```

2.6. PATCH

The PATCH method uses the HTTP PATCH method defined in [RFC5789] to provide an extensible framework for resource patching mechanisms. It is optional to implement by the server. Each patch type needs a unique media type. Zero or more PATCH media types MAY be supported by the server.

The "plain patch" PATCH method is used to create or update a sub-resource within the target resource. If the target resource instance does not exist, the server MUST NOT create it.

If the PATCH method succeeds, a "200 OK" Status-Line is returned if there is a message body, and "204 No Content" is returned if no response message body is sent.

If the user is not authorized to alter the target resource an error response containing a "403 Forbidden" Status-Line is returned to the client. All other error responses are handled according to the procedures defined in Section 6.

Example:

To replace just the "year" field in the "album" resource (instead of replacing the entire resource with the PUT method), 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
Content-Type: application/yang.data+json

{
  "example-jukebox:album" : {
    "genre" : "example-jukebox:rock",
    "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


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

2.7. DELETE

The DELETE method is used to delete the target resource. If the DELETE method succeeds, a "204 No Content" Status-Line is returned, and there is no response message body.

If the user is not authorized to delete the target resource then an error response containing a "403 Forbidden" Status-Line is returned to the client. All other error responses are handled according to the procedures defined in Section 6.

Example:

To delete a resource such as the "album" resource, the client might send:

```
DELETE /restconf/data/example-jukebox:jukebox/
   library/artist/Foo%20Fighters/album/Wasting%20Light HTTP/1.1
Host: example.com
```

If the resource is deleted, the server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:49:40 GMT
Server: example-server
```

2.8. Query Parameters

Each RESTCONF operation allows zero or more query parameters to be present in the request URI. The specific parameters that are allowed depends on the resource type, and sometimes the specific target resource used, in the request.

Name	Methods	Description
content	GET	Select config and/or non-config data resources
depth	GET	Request limited sub-tree depth in the reply content
filter	GET	Boolean notification filter for event-stream resources
insert	POST, PUT	Insertion mode for user-ordered data resources
point	POST, PUT	Insertion point for user-ordered data resources
select	GET	Request a subset of the target resource contents

start-time	GET	Replay buffer start time for event-stream resources
stop-time	GET	Replay buffer stop time for event-stream resources

RESTCONF Query Parameters

Query parameters can be given in any order. Each parameter can appear at most once in a request URI. A default value may apply if the parameter is missing.

The semantics and syntax for all query parameters are defined in the "query-parameters" YANG grouping in Section 7. The YANG encoding MUST be converted to URL-encoded string for use in the request URI.

Refer to Appendix D.3 for examples of query parameter usage.

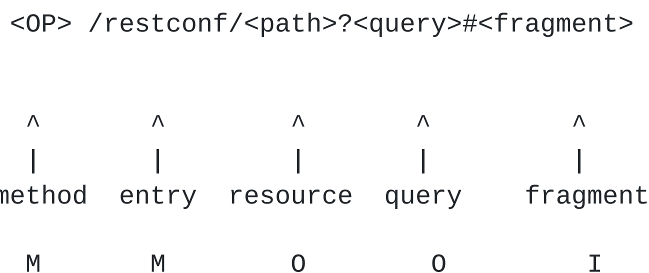
3. Messages

The RESTCONF protocol uses HTTP entities for messages. A single HTTP message corresponds to a single protocol method. Most messages can perform a single task on a single resource, such as retrieving a resource or editing a resource. The exception is the PATCH method, which allows multiple datastore edits within a single message.

3.1. Request URI Structure

Resources are represented with URIs following the structure for generic URIs in [RFC3986].

A RESTCONF operation is derived from the HTTP method and the request URI, using the following conceptual fields:



M=mandatory, O=optional, I=ignored

<text> replaced by client with real values

- o method: the HTTP method identifying the RESTCONF operation requested by the client, to act upon the target resource specified in the request URI. RESTCONF operation details are described in Section 2.

- o entry: the well-known RESTCONF entry point ("/restconf").

- o resource: the path expression identifying the resource that is being accessed by the operation. If this field is not present, then the target resource is the API itself, represented by the media type "application/yang.api".

- o query: the set of parameters associated with the RESTCONF message. These have the familiar form of "name=value" pairs. There is a specific set of parameters defined, although the server MAY choose to support additional parameters not defined in this document. The contents of the any query parameter value MUST be encoded according to [RFC2396], section 3.4. Any reserved characters MUST

be encoded with escape sequences, according to [RFC2396], section 2.4.

o fragment: This field is not used by the RESTCONF protocol.

When new resources are created by the client, a "Location" header is returned, which identifies the path of the newly created resource. The client MUST use this exact path identifier to access the resource once it has been created.

The "target" of an operation is a resource. The "path" field in the request URI represents the target resource for the operation.

3.2. Message Headers

There are several HTTP header lines utilized in RESTCONF messages. Messages are not limited to the HTTP headers listed in this section.

HTTP defines which header lines are required for particular circumstances. Refer to each operation definition section in Section 2 for examples on how particular headers are used.

There are some request headers that are used within RESTCONF, usually applied to data resources. The following tables summarize the headers most relevant in RESTCONF message requests:

Name	Description
Accept	Response Content-Types that are acceptable
Content-Type	The media type of the request body
Host	The host address of the server
If-Match	Only perform the action if the entity matches ETag
If-Modified-Since	Only perform the action if modified since time
If-Unmodified-Since	Only perform the action if un-modified since time

RESTCONF Request Headers

The following tables summarize the headers most relevant in RESTCONF message responses:

Name	Description
Allow	Valid actions when 405 error returned
Cache-Control	The cache control parameters for the response
Content-Type	The media type of the response body
Date	The date and time the message was sent
ETag	An identifier for a specific version of a resource
Last-Modified	The last modified date and time of a resource
Location	The resource identifier for a newly created resource

RESTCONF Response Headers

3.3. Message Encoding

RESTCONF messages are encoded in HTTP according to RFC 2616. The "utf-8" character set is used for all messages. RESTCONF message content is sent in the HTTP message body.

Content is encoded in either JSON or XML format.

XML encoding rules for data nodes are defined in [RFC6020]. The same encoding rules are used for all XML content.

JSON encoding rules are defined in [I-D.lhotka-netmod-json]. This encoding is valid JSON, but also has special encoding rules to identify module namespaces and provide consistent type processing of YANG data.

Request input content encoding format is identified with the Content-Type header. This field MUST be present if a message body is sent by the client.

Response output content encoding format is identified with the Accept header in the request, or if is not specified, the request input encoding format is used. If there was no request input, then the default output encoding is XML. File extensions encoded in the request are not used to identify format encoding.

3.4. RESTCONF Meta-Data

The RESTCONF protocol needs to retrieve the same meta-data that is used in the NETCONF protocol. Information about default leafs, last-modified timestamps, etc. are commonly used to annotate representations of the datastore contents. This meta-data is not

defined in the YANG schema because it applies to the datastore, and is common across all data nodes.

This information is encoded as attributes in XML, but JSON does not have a standard way of attaching non-schema defined meta-data to a resource.

3.4.1. JSON Encoding of RESTCONF Meta-Data

The YANG to JSON mapping [I-D.lhotka-netmod-json] does not support attributes because YANG does not support meta-data in data node definitions. This section specifies how RESTCONF meta-data is encoded in JSON.

Only simple meta-data is supported:

- o A meta-data instance can appear 0 or 1 times for a particular data node

- o A meta-data instance associated with a resource is encoded as if it were a YANG leaf of type "string", according to the encoding rules in [I-D.lhotka-netmod-json], except the identifier is prepended with a "@" (%40) character.

- o A meta-data instance associated with a YANG leaf or leaf-list within a resource is encoded as if it were a container for the meta-data values and the resource value in its native encoding. It is encoded according to the rules in [I-D.lhotka-netmod-json], except the meta-data identifiers are prepended with a "@" (%40) character. The resource name/value pair is repeated inside this container, which contains the actual value of the resource.

Example:

```
Meta-data:

  enabled=<boolean>
  owner=<owner-name>

YANG module: example
YANG example:

  container top {
    leaf A {
      type int32;
    }
    leaf B {
      type boolean;
    }
  }
```

The client is retrieving the "top" data resource, and the server is including datastore meta-data. Note that a query parameter to request or suppress specific meta-data is not provided in RESTCONF.

```
GET /restconf/data/example:top HTTP/1.1
Host: example.com
Accept: application/yang.data+json
```

The server might respond as follows:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Content-Type: application/yang.data+json
```



```
{
  "example:top": {
    "@enabled" : "true",
    "@owner" : "fred",
    "A" : {
      "@enabled" : "true",
      "A" : 42
    },
    "B" : {
      "@enabled" : "false",
      "B" : true
    }
  }
}
```

3.5. Return Status

Each message represents some sort of resource access. An HTTP "Status-Line" header line is returned for each request. If a 4xx or 5xx range status code is returned in the Status-Line, then the error information will be returned in the response, according to the format defined in Section 6.1.

3.6. Message Caching

Since the datastore contents change at unpredictable times, responses from a RESTCONF server generally SHOULD NOT be cached.

The server SHOULD include a "Cache-Control" header in every response that specifies whether the response should be cached. A "Pragma" header specifying "no-cache" MAY also be sent in case the "Cache-Control" header is not supported.

Instead of using HTTP caching, the client SHOULD track the "ETag" and/or "Last-Modified" headers returned by the server for the datastore resource (or data resource if the server supports it). A retrieval request for a resource can include the "If-None-Match" and/or "If-Modified-Since" headers, which will cause the server to return a "304 Not Modified" Status-Line if the resource has not changed. The client MAY use the HEAD method to retrieve just the message headers, which SHOULD include the "ETag" and "Last-Modified" headers, if this meta-data is maintained for the target resource.

4. Resources

The RESTCONF protocol operates on a hierarchy of resources, starting with the top-level API resource itself. Each resource represents a manageable component within the device.

A resource can be considered a collection of conceptual data and the set of allowed methods on that data. It can contain child nodes that are nested resources. The child resource types and methods allowed on them are data-model specific.

A resource has its own media type identifier, represented by the "Content-Type" header in the HTTP response message. A resource can contain zero or more nested resources. A resource can be created and deleted independently of its parent resource, as long as the parent resource exists.

All RESTCONF resources are defined in this document except datastore contents, protocol operations, and notification events. The syntax and semantics for these resource types are defined in YANG modules.

The RESTCONF resources are accessed via a set of URIs defined in this document. The set of YANG modules supported by the server will determine the additional data model specific operations, top-level data node resources, and notification event messages supported by the server.

The resources used in the RESTCONF protocol are identified by the "path" component in the request URI. Each operation is performed on a target resource.

4.1. RESTCONF Resource Types

The RESTCONF protocol defines some application specific media types to identify each of the available resource types. The following resource types are defined in RESTCONF:

Resource	Media Type
API	application/yang.api
Datastore	application/yang.datastore
Data	application/yang.data
Operation	application/yang.operation
Schema	application/yang
Stream	application/yang.stream

RESTCONF Media Types

4.2. Resource Discovery

A client SHOULD start by retrieving the top-level API resource, using the entry point URI "/restconf".

The RESTCONF protocol does not include a resource discovery mechanism. Instead, the definitions within the YANG modules advertised by the server are used to construct a predictable operation or data resource identifier.

The "depth" query parameter can be used to control how many descendant levels should be included when retrieving sub-resources. This parameter can be used with the GET method to discover sub-resources within a particular resource.

4.3. API Resource (/restconf)

The API resource contains the state and access points for the RESTCONF features. It is the top-level resource and has the media type "application/yang.api+xml" or "application/yang.api+json". It is accessible through the well-known URI "/restconf".

YANG Tree Diagram for "application/yang.api" Resource Type:

```
+--rw restconf
   +--rw data
   +--rw modules
   |  +--rw module [name revision]
   |     +--rw name         yang:yang-identifier
   |     +--rw revision     union
   |     +--rw schema?      empty
   |     +--rw namespace    inet:uri
   |     +--rw feature*     yang:yang-identifier
   |     +--rw deviation*   yang:yang-identifier
   |     +--rw submodule [name revision]
   |        +--rw name        yang:yang-identifier
   |        +--rw revision    union
   |        +--rw schema?     empty
   +--rw operations
   +--rw streams
   |  +--rw stream [name]
   |     +--rw name                         string
   |     +--rw description?                 string
   |     +--rw replay-support?              boolean
   |     +--rw replay-log-creation-time?   yang:date-and-time
   |     +--rw events?                      empty
```

```
      +--ro version?      enumeration
```

The "restconf" container definition in the "ietf-restconf" module defined in Section 7 is used to specify the structure and syntax of the conceptual sub-resources within the API resource.

This resource has the following child resources:

Child Resource	Description
data	Contains all data resources
modules	YANG module information
operations	Data-model specific operations
streams	Notification event streams
version	RESTCONF API version

RESTCONF Resources

4.3.1. /restconf/data

This mandatory resource represents the combined configuration and operational data resources that can be accessed by a client. It cannot be created or deleted by the client. The datastore resource type is defined in Section 4.4.

Example:

This example request by the client would retrieve only the non-configuration data nodes that exist within the "library" resource, using the "content" query parameter.

```
GET /restconf/data/example-jukebox:jukebox/library
   ?content=nonconfig  HTTP/1.1
Host: example.com
Accept: application/yang.data+json
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:30 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json

{
  "example-jukebox:library" : {
     "artist-count" : 42,
     "album-count" : 59,
     "song-count" : 374
  }
}
```

4.3.2. /restconf/modules

This mandatory resource contains the identifiers for the YANG data model modules supported by the server.

The server MUST maintain a last-modified timestamp for this resource, and return the "Last-Modified" header when this resource is retrieved with the GET or HEAD methods.

The server SHOULD maintain an entity-tag for this resource, and return the "ETag" header when this resource is retrieved with the GET or HEAD methods.

4.3.2.1. /restconf/modules/module

This mandatory resource contains one list entry for each YANG data model module supported by the server. There MUST be an instance of this resource for every YANG module that is accessible via an operation resource or a data resource.

The contents of the "module" resource are defined in the "module" YANG list statement in Section 7.

The server MAY maintain a last-modified timestamp for each instance of this resource, and return the "Last-Modified" header when this resource is retrieved with the GET or HEAD methods. If not supported then the timestamp for the parent "modules" resource MAY be used instead.

The server MAY maintain an entity-tag for each instance of this resource, and return the "ETag" header when this resource is retrieved with the GET or HEAD methods. If not supported then the timestamp for the parent "modules" resource MAY be used instead.

4.3.3. /restconf/operations

This optional resource is a container that provides access to the data-model specific protocol operations supported by the server. The server MAY omit this resource if no data-model specific operations are advertised.

Any data-model specific operations defined in the YANG modules advertised by the server MAY be available as child nodes of this resource.

Operation resources are defined in Section 4.6.

4.3.4. /restconf/streams

This optional resource is a container that provides access to the notification event streams supported by the server. The server MAY omit this resource if no notification event streams are supported. The media type for this resource is "application/yang.api".

The server will populate this container with a stream list entry for each stream type it supports. Each stream contains a leaf called "events" which represents an event stream resource. The media type for this resource is "application/yang.stream".

Stream resources are defined in Section 4.8. Notifications are defined in Section 5.

4.3.5. /restconf/version

This sub-resource can be used by the client to identify the exact version of the RESTCONF protocol implemented by the server. The same server-wide response MUST be returned each time this resource is retrieved.

The value is assigned by the server when the server is started. The server MUST return the value "1.0" for this version of the RESTCONF protocol. This resource is encoded with the rules for an "enumeration" data type, using the "version" leaf definition in Section 7.

4.4. Datastore Resource

The /restconf/data subtree represents the datastore resource type, which is a collection of configuration and operational data nodes.

A "unified datastore" interface is used to simplify resource editing for the client. The RESTCONF unified datastore is a conceptual

interface to the native configuration datastores that are present on the device.

The underlying NETCONF datastores (i.e., candidate, running, startup) can be used to implement the unified datastore, but the server design is not limited to the exact datastore procedures defined in NETCONF.

The "candidate" and "startup" datastores are not visible in the RESTCONF protocol. Transaction management and configuration persistence are handled by the server and not controlled by the client.

4.4.1. Edit Collision Detection

Two "edit collision detection" mechanisms are provided in RESTCONF, for datastore and data resources.

4.4.1.1. Timestamp

The last change time is maintained and the "Last-Modified" and "Date" headers are returned in the response for a retrieval request. The "If-Unmodified-Since" header can be used in edit operation requests to cause the server to reject the request if the resource has been modified since the specified timestamp.

The server MUST maintain a last-modified timestamp for this resource, and return the "Last-Modified" header when this resource is retrieved with the GET or HEAD methods. Only changes to configuration data resources within the datastore affect this timestamp.

4.4.1.2. Entity tag

A unique opaque string is maintained and the "ETag" header is returned in the response for a retrieval request. The "If-Match" header can be used in edit operation requests to cause the server to reject the request if the resource entity tag does not match the specified value.

The server MUST maintain a resource entity tag for this resource, and return the "ETag" header when this resource is retrieved with the GET or HEAD methods. The resource entity tag MUST be changed to a new previously unused value if changes to any configuration data resources within the datastore are made.

A datastore resource can only be written directly with the PATCH method. Only the configuration data resources within the datastore resource can be edited directly with all methods.]

Each RESTCONF edit of a datastore resource is saved to non-volatile storage in an implementation-specific matter by the server. There is no guarantee that configuration changes are saved immediately, or that the saved configuration is always a mirror of the running configuration.

4.5. Data Resource

A data resource represents a YANG data node that is a descendant node of a datastore resource.

For configuration data resources, the server MAY maintain a last-modified timestamp for the resource, and return the "Last-Modified" header when it is retrieved with the GET or HEAD methods. If maintained, the resource timestamp MUST be set to the current time whenever the resource or any configuration resource within the resource is altered.

For configuration data resources, the server MAY maintain a resource entity tag for the resource, and return the "ETag" header when it is retrieved as the target resource with the GET or HEAD methods. If maintained, the resource entity tag MUST be updated whenever the resource or any configuration resource within the resource is altered.

A data resource can be retrieved with the GET method. Data resources can be accessed via the "/restconf/data" entry point. This sub-tree is used to retrieve and edit data resources.

A configuration data resource can be altered by the client with some of all of the edit operations, depending on the target resource and the specific operation. Refer to Section 2 for more details on edit operations.

The resource definition version for a data resource is identified by the revision date of the YANG module containing the YANG definition for the data resource, specified in the /restconf/modules sub-tree.

4.5.1. Encoding YANG Instance Identifiers in the Request URI

In YANG, data nodes are named with an absolute XPath expression, defined in [XPath] , starting from the document root to the target resource. In RESTCONF, URL encoded Location header expressions are used instead.

The YANG "instance-identifier" (i-i) data type is represented in RESTCONF with the path expression format defined in this section.

Name	Comments
point	Insertion point is always a full i-i
path	Request URI path is a full or partial i-i

RESTCONF instance-identifier Type Conversion

The "path" component of the request URI contains the absolute path expression that identifies the target resource.

A predictable location for a data resource is important, since applications will code to the YANG data model module, which uses static naming and defines an absolute path location for all data nodes.

A RESTCONF data resource identifier is not an XPath expression. It is encoded from left to right, starting with the top-level data node, according to the "api-path" rule in Section 4.5.1.1. The node name of each ancestor of the target resource node is encoded in order, ending with the node name for the target resource.

If a data node in the path expression is a YANG list node, then the key values for the list (if any) are encoded according to the "key-value" rule. If the list node is the target resource, then the key values MAY be omitted, according to the operation. For example, the POST method to create a new data resource for a list node does not require key values to be present in the request URI.

The key leaf values for a data resource representing a YANG list MUST be encoded as follows:

- The value of each leaf identified in the "key" statement is encoded in order.

- All the components in the "key" statement MUST be encoded. Partial instance identifiers are not supported.

- Each value is encoded using the "key-value" rule in Section 4.5.1.1, according to the encoding rules for the data type of the key leaf.

- An empty string can be a valid key value (e.g., "/top/list/key1//key3").

- The "/" character MUST be URL-encoded (i.e., "%2F").

- All whitespace MUST be URL-encoded.

- A "null" value is not allowed since the "empty" data type is not allowed for key leafs.

- The XML encoding is defined in [RFC6020].

- The JSON encoding is defined in [I-D.lhotka-netmod-json].

- The entire "key-value" MUST be properly URL-encoded, according to the rules defined in [RFC3986].

- resource URI values returned in Location headers for data resources MUST identify the module name, even if there are no conflicting local names when the resource is created. This ensures the correct resource will be identified even if the server loads a new module that the old client does not know about.

Examples:

```
[ lines wrapped for display purposes only ]

/restconf/data/example-jukebox:jukebox/library/
   artist/Beatles/album

/restconf/data/example-list:newlist/17
   /nextlist%2Ffoo%2Fbar%2Facme-list-ext%3Aext-leaf

/restconf/data/example-list:somelist/the%20key

/restconf/data/example-list:somelist/the%20key/address
```

4.5.1.1. ABNF For Data Resource Identifiers

The following ABNF syntax is used to construct RESTCONF path identifiers:

```
api-path = "/"  |
           ("/" api-identifier
             0*("/" (api-identifier | key-value )))

api-identifier = [module-name ":"] identifier

module-name = identifier

key-value = string

;; An identifier MUST NOT start with
;; (('X'|'x') ('M'|'m') ('L'|'l'))
identifier  = (ALPHA / "_")
              *(ALPHA / DIGIT / "_" / "-" / ".")

string = <an unquoted string>

[FIXME: the syntax for the select string is still TBD]
api-select = api-identifier
                0*("/" (api-identifier | key-value ))
```

4.5.2. Defaults Handling

NETCONF has a rather complex model for handling default values for leafs. RESTCONF attempts to avoid this complexity by restricting the operations that can be applied to a resource.

If the target of a GET method is a data node that represents a leaf that has a default value, and the leaf has not been given a value yet, the server MUST return the default value that is in use by the server.

If the target of a GET method is a data node that represents a container or list that has any sub-resources with default values, for the sub-resources that have not been given value yet, the server MAY return the default values that are in use by the server.

4.6. Operation Resource

An operation resource represents an protocol operation defined with the YANG "rpc" statement.

All operation resources share the same module namespace as any top-level data resources, so the name of an operation resource cannot conflict with the name of a top-level data resource defined within the same module.

If 2 different YANG modules define the same "rpc" identifier, then

the module name MUST be used in the request URI. For example, if "module-A" and "module-B" both defined a "reset" operation, then invoking the operation from "module-A" would be requested as follows:

```
POST /restconf/operations/module-A:reset HTTP/1.1
Server example.com
```

Any usage of an operation resource from the same module, with the same name, refers to the same "rpc" statement definition. This behavior can be used to design protocol operations that perform the same general function on different resource types.

If the "rpc" statement has an "input" section, then a message body MAY be sent by the client in the request, otherwise the request message MUST NOT include a message body. If the "rpc" statement has an "output" section, then a message body MAY be sent by the server in the response. Otherwise the server MUST NOT include a message body in the response message, and MUST send a "204 No Content" Status-Line instead.

4.6.1. Encoding Operation Input Parameters

If the "rpc" statement has an "input" section, then the "input" node is provided in the message body, corresponding to the YANG data definition statements within the "input" section.

Example:

The following YANG definition is used for the examples in this section.

```
rpc reboot {
  input {
    leaf delay {
      units seconds;
      type uint32;
      default 0;
    }
    leaf message { type string; }
    leaf language { type string; }
  }
}
```

The client might send the following POST request message:

```
POST /restconf/operations/example-ops:reboot HTTP/1.1
Host: example.com
Content-Type: application/yang.operation+json

{
  "example-ops:input" : {
    "delay" : 600,
    "message" : "Going down for system maintenance",
    "language" : "en-US"
  }
}
```

The server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 25 Apr 2012 11:01:00 GMT
Server: example-server
```

4.6.2. Encoding Operation Output Parameters

If the "rpc" statement has an "output" section, then the "output" node is provided in the message body, corresponding to the YANG data definition statements within the "output" section.

Example:

The following YANG definition is used for the examples in this section.

```
rpc get-reboot-info {
  output {
    leaf reboot-time {
      units seconds;
      type uint32;
    }
    leaf message { type string; }
    leaf language { type string; }
  }
}
```

The client might send the following POST request message:

```
POST /restconf/operations/example-ops:get-reboot-info HTTP/1.1
Host: example.com
Accept: application/yang.operation+json
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 25 Apr 2012 11:10:30 GMT
Server: example-server
Content-Type: application/yang.operation+json
```


```
{
  "example-ops:output" : {
    "reboot-time" : 30,
    "message" : "Going down for system maintenance",
    "language" : "en-US"
  }
}
```


4.7. Schema Resource

If the server supports the "schema" leaf within the API then the client can retrieve the YANG schema text for the associated YANG module or submodule, using the GET method.

The client might send the following GET request message:

```
GET /restconf/modules/module/example-jukebox/2013-12-21/schema
   HTTP/1.1
Host: example.com
Accept: application/yang
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 25 Apr 2012 11:10:30 GMT
Server: example-server
Content-Type: application/yang

module example-jukebox {

   namespace "http://example.com/ns/example-jukebox";
   prefix "jbox";

   // rest of YANG module content deleted...
}
```

4.8. Stream Resource

A stream resource represents a source for system generated event notifications. Each stream is created and modified by the server only. A client can retrieve a stream resource or initiate a long-poll server sent event stream, using the procedure specified in Section 5.3.

A notification stream functions according to the NETCONF Notifications specification [RFC5277]. The "ietf-restconf" YANG module contains the "stream" list (/restconf/streams/stream) which specifies the syntax and semantics of a stream resource.

5. Notifications

The RESTCONF protocol supports YANG-defined event notifications. The solution preserves aspects of NETCONF Event Notifications [RFC5277] while utilizing the Server-Sent Events [wd-eventsource] transport strategy.

5.1. Server Support

A RESTCONF server is not required to support RESTCONF notifications. Clients may determine if a server supports RESTCONF notifications by using the HTTP operation OPTIONS, HEAD, or GET on the "/restconf/streams" resource described below. The server does not support RESTCONF notifications if an HTTP error code is returned (e.g. 404 Not Found).

5.2. Event Streams

A RESTCONF server that supports notifications will populate a stream resource for each notification delivery service access point. A RESTCONF client can retrieve the list of supported event streams from a RESTCONF server using the GET operation on the "/restconf/streams" resource.

The "/restconf/streams" container definition in the "ietf-restconf" module defined in Section 7 is used to specify the structure and syntax of the conceptual sub-resources within the "streams" resource.

For example:

The client might send the following request:

```
GET /restconf/streams HTTP/1.1
Host: example.com
Accept: application/yang.api+xml
```

The server might send the following response:

```
HTTP/1.1 200 OK
Content-Type: application/yang.api+xml
```

```
<streams xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
   <stream>
      <name>NETCONF</name>
      <description>default NETCONF event stream
      </description>
      <replay-support>true</replay-support>
      <replay-log-creation-time>
         2007-07-08T00:00:00Z
      </replay-log-creation-time>
      <events/>
   </stream>
   <stream>
      <name>SNMP</name>
      <description>SNMP notifications</description>
      <replay-support>false</replay-support>
      <events/>
   </stream>
   <stream>
      <name>syslog-critical</name>
      <description>Critical and higher severity
      </description>
      <replay-support>true</replay-support>
      <replay-log-creation-time>
         2007-07-01T00:00:00Z
      </replay-log-creation-time>
      <events/>
   </stream>
</streams>
```

5.3. Subscribing to Receive Notifications

RESTCONF clients can subscribe to receive notifications by sending an HTTP GET request for the "/restconf/streams/stream/<stream-name>" resource, with the "Accept" type "text/event-stream". The server will treat the connection as an event stream, using the Server Sent Events [wd-eventsource] transport strategy.

The server MAY support query parameters for a GET method on this resource. These parameters are specific to each notification stream.

For example:

```
GET /restconf/streams/stream/NETCONF/events HTTP/1.1
Host: example.com
Accept: text/event-stream
Cache-Control: no-cache
Connection: keep-alive
```

A RESTCONF client MAY request the server compress the events using the HTTP header field "Accept-Encoding". For instance:

```
GET /restconf/streams/stream/NETCONF/events HTTP/1.1
Host: example.com
Accept: text/event-stream
Cache-Control: no-cache
Connection: keep-alive
Accept-Encoding: gzip, deflate
```

5.3.1. NETCONF Event Stream

The server SHOULD support the "NETCONF" notification stream defined in [RFC5277]. For this stream, RESTCONF notification subscription requests MAY specify parameters indicating the events it wishes to receive.

Name	Description
start-time	replay event start time
stop-time	replay event stop time
filter	boolean content filter

NETCONF Stream Query Parameters

The semantics and syntax for these query parameters are defined in the "query-parameters" YANG grouping in Section 7. The YANG encoding MUST be converted to URL-encoded string for use in the request URI.

Refer to Appendix D.3.3 for filter parameter examples.

5.4. Receiving Event Notifications

RESTCONF notifications are encoded according to the definition of the event stream. The NETCONF stream defined in [RFC5277] is encoded in XML format.

The structure of the event data is based on the "notification" element definition in section 4 of [RFC5277]. It MUST conform to the "notification" YANG container definition in Section 7.

An example SSE notification encoded using XML:

```
   data: <notification
   data:    xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
   data:    <event-time>2013-12-21T00:01:00Z</event-time>
   data:    <event xmlns="http://example.com/event/1.0">
   data:       <eventClass>fault</eventClass>
   data:       <reportingEntity>
   data:           <card>Ethernet0</card>
   data:       </reportingEntity>
   data:       <severity>major</severity>
   data:     </event>
   data: </notification>
```

Since XML is not whitespace sensitive, the above message can be encoded onto a single line.

For example: ('\' line wrapping added for formatting only)

```
   data: <notification xmlns="urn:ietf:params:xml:ns:yang:ietf-rest\
   conf"><event-time>2013-12-21T00:01:00Z</event-time><event xmlns="\
   http://example.com/event/1.0"><eventClass>fault</eventClass><repo\
   rtingEntity><card>Ethernet0</card></reportingEntity><severity>maj\
   or</severity></event></notification>
```

The SSE specifications supports the following additional fields: event, id and retry. A RESTCONF server MAY send the "retry" field and, if it does, RESTCONF clients SHOULD use it. A RESTCONF server SHOULD NOT send the "event" or "id" fields, as there are no meaningful values that could be used for them that would not be redundant to the contents of the notification itself. RESTCONF servers that do not send the "id" field also do not need to support the HTTP header "Last-Event-Id". RESTCONF servers that do send the "id" field MUST still support the "startTime" query parameter as the preferred means for a client to specify where to restart the event stream.

6. Error Reporting

HTTP Status-Lines are used to report success or failure for RESTCONF operations. The <rpc-error> element returned in NETCONF error responses contains some useful information. This error information is adapted for use in RESTCONF, and error information is returned for "4xx" class of status codes.

The following table summarizes the return status codes used specifically by RESTCONF operations:

Status-Line	Description
100 Continue	POST accepted, 201 should follow
200 OK	Success with response body
201 Created	POST to create a resource success
202 Accepted	POST to create a resource accepted
204 No Content	Success without response body
304 Not Modified	Conditional operation not done
400 Bad Request	Invalid request message
403 Forbidden	Access to resource denied
404 Not Found	Resource target or resource node not found
405 Method Not Allowed	Method not allowed for target resource
409 Conflict	Resource or lock in use
412 Precondition Failed	Conditional method is false
413 Request Entity Too Large	too-big error
414 Request-URI Too Large	too-big error
415 Unsupported Media Type	non RESTCONF media type
500 Internal Server Error	operation-failed
501 Not Implemented	unknown-operation
503 Service Unavailable	Recoverable server error

HTTP Status Codes used in RESTCONF

Since an operation resource is defined with a YANG "rpc" statement, a mapping between the NETCONF <error-tag> value and the HTTP status code is needed. The specific error condition and response code to use are data-model specific and might be contained in the YANG "description" statement for the "rpc" statement.

<error-tag>	status code
in-use	409
invalid-value	400
too-big	413
missing-attribute	400
bad-attribute	400
unknown-attribute	400
bad-element	400
unknown-element	400
unknown-namespace	400
access-denied	403
lock-denied	409
resource-denied	409
rollback-failed	500
data-exists	409
data-missing	409
operation-not-supported	501
operation-failed	500
partial-operation	500
malformed-message	400

Mapping from error-tag to status code

6.1. Error Response Message

When an error occurs for a request message on a data resource or an operation resource, and a "4xx" class of status codes (except for status code "403"), then the server SHOULD send a response body containing the information described by the "errors" container definition within the YANG module Section 7.

YANG Tree Diagram for <errors> Data:

```
+--ro errors
   +--ro error
      +--ro error-type       enumeration
      +--ro error-tag        string
      +--ro error-app-tag?   string
      +--ro (error-node)?
      |  +--:(error-path)
      |  |  +--ro error-path?      instance-identifier
      |  +--:(error-urlpath)
      |     +--ro error-urlpath?   data-resource-identifier
      +--ro error-message?   string
      +--ro error-info
```

The semantics and syntax for RESTCONF error messages are defined in the "errors" YANG grouping in Section 7.

Examples:

The following example shows an error returned for an "lock-denied" error on a datastore resource.

```
POST /restconf/operations/example-ops:lock-datastore HTTP/1.1
Host: example.com
```

The server might respond:

```
HTTP/1.1 409 Conflict
Date: Mon, 23 Apr 2012 17:11:00 GMT
Server: example-server
Content-Type: application/yang.api+json

{
  "ietf-restconf:errors": {
    "error": {
      "error-type": "protocol",
      "error-tag": "lock-denied",
      "error-message": "Lock failed, lock already held"
    }
  }
}
```

The following example shows an error returned for a "data-exists" error on a data resource. The "jukebox" resource already exists so it cannot be created.

The client might send:

```
POST /restconf/data/example-jukebox:jukebox HTTP/1.1
Host: example.com
```

The server might respond:

```
HTTP/1.1 409 Conflict
Date: Mon, 23 Apr 2012 17:11:00 GMT
Server: example-server
Content-Type: application/yang.api+json
```


```
{
  "ietf-restconf:errors": {
    "error": {
      "error-type": "protocol",
      "error-tag": "data-exists",
      "error-urlpath": "http://example.com/restconf/data/
           example-jukebox:jukebox",
      "error-message":
        "Data already exists, cannot create new resource"
    }
  }
}
```

7. RESTCONF module

The "ietf-restconf" 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.

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

<CODE BEGINS> file "ietf-restconf@2014-02-13.yang"

```
module ietf-restconf {
  namespace "urn:ietf:params:xml:ns:yang:ietf-restconf";
  prefix "rc";

  import ietf-yang-types { prefix yang; }
  import ietf-inet-types { prefix inet; }

  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>";

  description
    "This module contains conceptual YANG specifications
```

```
        for the message and error content that is used in
        RESTCONF protocol messages. A conceptual container
        representing the RESTCONF API nodes is also defined
        for the media type application/yang.api.

        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) 2013 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-bierman-netconf-restconf-04.txt

     // RFC Ed.: update the date below with the date of RFC publication
     // and remove this note.
     revision 2014-02-13 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: RESTCONF Protocol.";
     }

     typedef data-resource-identifier {
       type string {
         length "1 .. max";
       }
       description
         "Contains a Data Resource Identifier formatted string
          to identify a specific data resource instance.
          The document root for all data resources is a
          datastore resource container. Each top-level YANG
          data nodes supported by the server will be represented
```

```
          as a child node of the document root.

          The canonical representation of a data resource identifier
          includes the full server specification that is returned
          in the Location header when a new data resource is created
          with the POST method.

          The abbreviated representation does not contain any server
          location identification. Instead the identifier will start
          with the '/' character to represent the datastore document
          root for the data resource instance.

          The server MUST accept either representation and SHOULD
          return the canonical representation in any response message.";
       reference
         "RFC XXXX: [sec. 5.3.1.1 ABNF For Data Resource Identifiers]";
     }

     typedef revision-identifier {
       type string {
         pattern '\d{4}-\d{2}-\d{2}';
       }
       description
         "Represents a specific date in YYYY-MM-DD format.
          TBD: make pattern more precise to exclude leading zeros.";
     }

     grouping errors {

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

       container errors {
         config false;  // needed so list error does not need a key
         description
           "Represents an error report returned by the server if
            a request results in an error.";

         list error {
           description
             "An entry containing information about one
              specific error that occurred while processing
              a RESTCONF request.";
           reference "RFC 6241, Section 4.3";

           leaf error-type {
```

```
             type enumeration {
               enum transport {
                 description "The transport layer";
               }
               enum rpc {
                 description "The rpc or notification layer";
               }
               enum protocol {
                 description "The protocol operation layer";
               }
               enum application {
                 description "The server application layer";
               }
             }
             mandatory true;
             description
               "The protocol layer where the error occurred.";
           }

           leaf error-tag {
             type string;
             mandatory true;
             description
               "The enumerated error tag.";
           }

           leaf error-app-tag {
             type string;
             description
               "The application-specific error tag.";
           }

           choice error-node {
             description
               "The server will return the location of the error node
                in a format that is appropriate for the protocol.
                If no specific node within the request message body
                caused the error then this choice will not be present.";

             leaf error-path {
               type instance-identifier;
               description
                 "The YANG instance identifier associated
                  with the error node. This leaf will only be
                  present if the error node is not a data resource,
                  e.g., the error node is an input parameter
                  for an operation resource.";
             }
```

```
            leaf error-urlpath {
              type data-resource-identifier;
              description
                "The target data resource identifier associated
                 with the error node.  This leaf will only be
                 present if the error node is associated with
                 a data resource (either within the server or
                 in the request message).";
            }
          }

          leaf error-message {
            type string;
            description
              "A message describing the error.";
          }

          anyxml error-info {
             description
               "Arbitrary XML that represents a container
                of additional information for the error report.";
          }
        }
      }
    } // grouping errors


    grouping restconf {
      description
        "A grouping that contains a YANG container
         representing the syntax and semantics of
         the RESTCONF API resource.";

      container restconf {
        description
          "Conceptual container representing the
           application/yang.api resource type.";

        container data {
          description
            "Container representing the application/yang.datastore
             resource type. Represents the conceptual root of all
             operational data and configuration data supported by
             the server.  The child nodes of this container can be
             any data resource (application/yang.data), which are
             defined as top-level data nodes from the YANG modules
             advertised by the server in /restconf/modules.";
        }
```

```
     container modules {
       description
         "Contains a list of module description entries.
          These modules are currently loaded into the server.";

       grouping common-leafs {
         description
           "Common parameters for YANG modules and submodules.";

         leaf name {
           type yang:yang-identifier;
           description "The YANG module or submodule name.";
         }
         leaf revision {
           type union {
             type revision-identifier;
             type string { length 0; }
           }
           description
             "The YANG module or submodule revision date.
              An empty string is used if no revision statement
              is present in the YANG module or submodule.";
         }

         leaf schema {
           type empty;
           description
             "Represents the YANG schema resource for this module
              or submodule if it is available on the server.
              This leaf will only be present if the server has
              the schema available for retrieval.  A GET
              request with a target resource URI that identifies
              this leaf will cause the server to return the YANG
              schema text for the associated module or submodule.";
         }
       }


       list module {
         key "name revision";
         description
           "Each entry represents one module currently
            supported by the server.";

         uses common-leafs;

         leaf namespace {
           type inet:uri;
```

```
            mandatory true;
            description
              "The XML namespace identifier for this module.";
          }
          leaf-list feature {
            type yang:yang-identifier;
            description
              "List of YANG feature names from this module that are
               supported by the server.";
          }
          leaf-list deviation {
            type yang:yang-identifier;
            description
              "List of YANG deviation module names used by this
               server to modify the conformance of the module
               associated with this entry.";
          }

          list submodule {
            key "name revision";
            description
              "Each entry represents one submodule within the
               parent module.";

            uses common-leafs;
          }
        }
      }

      container operations {
        description
          "Container for all operation resources
           (application/yang.operation),

           Each resource is represented as an empty leaf with the
           name of the RPC operation from the YANG rpc statement.

           E.g.;

              POST /restconf/operations/show-log-errors

              leaf show-log-errors {
                type empty;
              }
          ";
      }

      container streams {
```

```
          description
            "Container representing the notification event streams
             supported by the server.";
           reference
             "RFC 5277, Section 3.4, <streams> element.";

          list stream {
            key name;
            description
              "Each entry describes an event stream supported by
               the server.";

            leaf name {
              type string;
              description "The stream name";
              reference "RFC 5277, Section 3.4, <name> element.";
            }

            leaf description {
              type string;
              description "Description of stream content";
              reference
                "RFC 5277, Section 3.4, <description> element.";
            }

            leaf replay-support {
              type boolean;
              description
                "Indicates if replay buffer supported for this stream";
              reference
                "RFC 5277, Section 3.4, <replaySupport> element.";
            }

            leaf replay-log-creation-time {
              type yang:date-and-time;
              description
                "Indicates the time the replay log for this stream
                 was created.";
              reference
                "RFC 5277, Section 3.4, <replayLogCreationTime>
                 element.";
            }

            leaf events {
              type empty;
              description
                "Represents the entry point for establishing
                 notification delivery via server sent events.";
```

```
             }
           }
         }

         leaf version {
           type enumeration {
             enum "1.0" {
               description
                 "Version 1.0 of the RESTCONF protocol.";
             }
           }
           config false;
           description
             "Contains the RESTCONF protocol version.";
         }
       }
     }  // grouping restconf


     grouping query-parameters {
       description
          "Contains conceptual definitions for the query string
           parameters used in the RESTCONF protocol.";

       leaf content {
         type enumeration {
           enum config {
             description
               "Return only configuration descendant data nodes";
           }
           enum nonconfig {
             description
               "Return only non-configuration descendant data nodes";
           }
           enum all {
             description
               "Return all descendant data nodes";
           }
         }
         description
            "The content parameter controls how descendant nodes of
             the requested data nodes will be processed in the reply.

             This parameter is only allowed for GET methods on
             datastore and data resources.  A 400 Bad Request error
             is returned if used for other methods or resource types.

             The default value is determined by the config-stmt
```

```
             value of the requested data nodes. If 'false', then
             the default is 'nonconfig'.  If 'true' then the
             default is 'config'.";
       }

       leaf depth {
         type union {
           type enumeration {
             enum unbounded {
               description "All sub-resources will be returned.";
             }
           }
           type uint32 {
             range "1..max";
           }
         }
         default unbounded;
         description
           "The 'depth' parameter is used to specify the number
            of nest levels returned in a response for a GET method.
            The first nest-level consists of the requested data node
            itself. Any child nodes which are contained within
            a parent node have a depth value that is 1 greater than
            its parent.

            This parameter is only allowed for GET methods on api,
            datastore, and data resources.  A 400 Bad Request error
            is returned if used for other methods or resource types.

            By default, the server will include all sub-resources
            within a retrieved resource, which have the same resource
            type as the requested resource. Only one level of
            sub-resources with a different media type than the target
            resource will be returned.";
       }

       leaf filter {
         type yang:xpath1.0;
         description
           "The 'filter' parameter is used to indicate which subset of
            all possible events are of interest.  If not present, all
            events not precluded by other parameters will be sent.

            This parameter is only allowed for GET methods on a
            text/event-stream data resource.  A 400 Bad Request error
            is returned if used for other methods or resource types.

            The format of this parameter is an XPath expression, and
```

```
            is evaluated in the following context:

            o  The set of namespace declarations is the set of
               prefix and namespace pairs for all supported YANG
               modules, where the prefix is the YANG module name, and
               the namespace is as defined by the 'namespace' statement
               in the YANG module.
            o  The function library is the core function library defined
               in XPATH.

            o  The set of variable bindings is empty.

            o  The context node is the root node

           The filter is used as defined in [RFC5277], section 3.6.
           If the boolean result of the expression is true when applied
           to the conceptual 'notification' document root, then the
           notification event is delivered to the client.";
       }

       leaf insert {
         type enumeration {
           enum first {
             description "Insert the new data as the new first entry.";
           }
           enum last {
             description "Insert the new data as the new last entry.";
           }
           enum before {
             description
               "Insert the new data before the insertion point,
                specified by the value of the 'point' parameter.";
           }
           enum after {
             description
               "Insert the new data after the insertion point,
                specified by the value of the 'point' parameter.";
           }
         }
         default last;
         description
           "The 'insert' parameter is used to specify how a
            resource should be inserted within a user-ordered list.

            This parameter is only supported for the POST and PUT
            methods. It is also only supported if the target
            resource is a data resource, and that data represents
            a YANG list or leaf-list that is ordered by the user.
```

```
          If the values 'before' or 'after' are used,
          then a 'point' query parameter for the insertion
          parameter MUST also be present, or a 400 Bad Request
          error is returned.";
     }

     leaf point {
       type data-resource-identifier;
       description
         "The 'point' parameter is used to specify the
          insertion point for a data resource that is being
          created or moved within a user ordered list or leaf-list.

          This parameter is only supported for the POST and PUT
          methods. It is also only supported if the target
          resource is a data resource, and that data represents
          a YANG list or leaf-list that is ordered by the user.

          If the 'insert' query parameter is not present, or has
          a value other than 'before' or 'after', then a 400
          Bad Request error is returned.

          This parameter contains the instance identifier of the
          resource to be used as the insertion point for a
          POST or PUT method.";
     }

     leaf select {
       type string {
         length "1 .. max";
       }
       description
         "The 'select' query parameter is used to specify an
          expression which can represent a subset of all data nodes
          within the target resource. It contains an expression
          string, using the target resource as the context node.

          The encoding for the select parameter is still TBD.

          This parameter is only allowed for GET methods on api,
          datastore, and data resources.  A 400 Bad Request error
          is returned if used for other methods or resource types.

          If XPath:
          The string is an XPath expression that will be evaluated
          using the target resource instance as the context node
          and the document root. It is expected to return a node-set
          result representing the descendants within the context
```

```
            node that should be returned in a GET response.";
       }

       leaf start-time {
         type yang:date-and-time;
         description
           "The 'start-time' parameter is used to trigger
            the notification replay feature and indicate
            that the replay should start at the time specified.
            If the stream does not support replay, per the
            'replay-support' attribute returned by
            the /restconf/streams resource, then the server MUST
            return the HTTP error code 400 Bad Request.

            This parameter is only allowed for GET methods on a
            text/event-stream data resource.  A 400 Bad Request error
            is returned if used for other methods or resource types.

            If this parameter is not present, then a replay subscription
            is not begin requested.  It is not valid to specify start
            times that are later than the current time.  If the value
            specified is earlier than the log can support, the replay
            will begin with the earliest available notification";
       }

       leaf stop-time {
         type yang:date-and-time;
         description
           "The 'stop-time' parameter is used with the
            replay feature to indicate the newest notifications of
            interest.  This parameter MUST be used with and have a
            value later than the 'start-time' parameter.

            This parameter is only allowed for GET methods on a
            text/event-stream data resource.  A 400 Bad Request error
            is returned if used for other methods or resource types.

            If this parameter is not present, the notifications will
            continue until the subscription is terminated.
            Values in the future are valid.";
       }

     }  // grouping query-parameters


     grouping notification {
       description
         "Contains the notification message wrapper definition.";
```

```
      container notification {
        description
          "RESTCONF notification message wrapper.";

        leaf event-time {
          type yang:date-and-time;
          mandatory true;
          description
            "The time the event was generated by the
             event source.";
          reference
            "RFC 5277, section 4, <eventTime> element.";
        }

        /* The YANG-specific notification container is encoded
         * after the 'event-time' element.  The format
         * corresponds to the notificationContent element
         * in RFC 5277, section 4. For example:
         *
         *  module example-one {
         *     ...
         *     notification event1 { ... }
         *
         *  }
         *
         *  Encoded as element 'event1' in the namespace
         *  for module 'example-one'.
         */
      }
    }  // grouping notification

  }

  <CODE ENDS>
```

8. IANA Considerations

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

8.2. application/yang Media Sub Types

The parent MIME media type for RESTCONF resources is application/yang, which is defined in [RFC6020]. This document defines the following sub-types for this media type.

- api
- data
- datastore
- operation
- stream

Type name: application

Subtype name: yang.xxx

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

9. Security Considerations

TBD

10. References

10.1. Normative References

[I-D.lhotka-netmod-json]
Lhotka, L., "Modeling JSON Text with YANG", draft-lhotka-netmod-yang-json-02 (work in progress), September 2013.

[JSON] Bray, T., Ed., "The JSON Data Interchange Format", draft-ietf-json-rfc4627bis-10 (work in progress), December 2013.

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

[RFC2396] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[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.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, July 2008.

[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.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012.

[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>.

[wd-eventsource]
Hickson, I., "Server-Sent Events", December 2012.

10.2. Informative References

[XPath] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>.

Appendix A. Change Log

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

A.1. 03 to 04

- Changed term RESTful to REST-like
- Clarified SSE section
- Clarified access control section
- Clarified return code test for some HTTP methods
- Fixed 2 examples that were wrong

A.2. 02 to 03

- Move YANG Patch to separate document
- Remove some introduction text
- Move most framework text into other sections
- Move notifications into its own section
- Remove fields; all YANG node types are sub-resources
- Add schema retrieval via /restconf/modules/module/schema resource
- Add submodule support to /restconf/modules/module resource
- Move some examples to appendix
- Add canonical representation for data-resource-identifier typedef
- Move query string definitions to YANG module
- Removed "format" query parameter
- Removed user-selected XML/JSON encoding for events. Event streams define a fixed encoding. It is not user-configurable.
- Clarified that query parameters for SSE event streams are specific to the stream definition; start-time, stop-time, and filter only apply to the NETCONF stream;

- o Change operation input and output media type from application/ yang.data to application/yang.data.

- o Fix some bugs and typos

A.3. 01 to 02

- o Added Notification Model (section 2.2)

- o Remove error-action from YANG Patch

- o Add "comment" and "ok" leafs to yang-patch-status container

- o Fixed YANG Patch JSON example syntax

- o Added stream resource type and streams container to /restconf container

- o Removed "vnd" from media type definitions

- o Changed yang-patch edit list from ascending uint32 key to an arbitrary string key and an ordered-by user list.

- o Several clarifications and corrections

- o Add YANG tree diagrams

- o Add application/yang.patch-status media type

- o Remove redundant "global-errors" container from "yang-patch-status" container

- o Split the /restconf/datastore entry point into 2 entry points (config and operational)

- o Remove the "config" parameter since it is no longer needed after datastore is split

A.4. 00 to 01

- o Removed incorrect /.well-known URI prefix.

- o Remove incorrect IANA request for well-known URI.

- o Clarified that API resource type nodes are defined in the ietf-restconf namespace.

- Changed CamelCase names in example-jukebox.yang to lowercase, and updated examples.

- Updated and corrected YANG types in ietf-restconf module.

A.5. YANG-API-01 to RESTCONF-00

- Protocol renamed from YANG-API to RESTCONF

- Fields are clarified. Containers and lists are sub-resources. All other YANG data node types are fields within a parent resource.

- The 'optional-key' YANG extension has been removed.

- The default value is returned by the server if the target resource represents a missing data node but the server is using a default value for the leaf.

- The default for the 'depth' parameter has been changed from '1' to 'unbounded'. The depth is only limited if an integer value for this parameter is specified by the client.

- The default for the 'format' parameter has been changed from 'json' to 'xml'.

- expanded introduction

- removed transactions

- removed capabilities

- removed usage of Range and IfRange headers

- simplified editing model

- removed global protocol operations from ietf-restconf.yang

- changed RPC operation terminology to protocol operation

- updated JSON draft reference

- updated IANA section

- added YANG Patch

- added YANG definitions to ietf-restconf.yang

- added Kent Watsen and Rex Fernando as co-authors

- updated YANG modules so they pass pyang --ietf checking

- changed examples so resource URIs use the module name variant to identify data resources

- changed depth behavior so the entire server contents are not returned for "GET /restconf"; Server will stop at new resource type; e.g. yang.api --> yang.datastore returns the datastore as an empty node; yang.api --> yang.operation returns the operation name as an empty node;

Appendix B. Open Issues

B.1. message-id

- There is no "message-id" field in a RESTCONF message. Is a message identifier needed? If so, should either the "Message-ID" or "Content-ID" header from RFC 2392 be used for this purpose?

B.2. select parameter

- What syntax should be used for the "select" query parameter? The current choices are "XPath" and "path-expr". Perhaps an additional parameter to identify the select string format is needed to allow extensibility?

B.3. server support verification

- Are all header lines used by RESTCONF supported by common application frameworks, such as FastCGI and WSGI? If not, then should query parameters be used instead, since the QUERY_STRING is widely available to WEB applications?

B.4. error media type

- Should the <errors> element returned in error responses be a separate media type?

B.5. additional datastores

- How should additional datastores be supported, which may be added to the NETCONF/NETMOD framework in the future?

B.6. PATCH media type discovery

- How does a client know which PATCH media types are supported by the server in addition to application/yang.data and application/yang.patch?

B.7. RESTCONF version

- Is the /restconf/version field considered meta-data? Should it be returned as XRD (Extensible Resource Descriptor)? In addition or instead of the version field? Should this be the ietf-restconf YANG module revision date, instead of the string 1.0?

B.8. YANG to resource mapping

- Since data resources can only be YANG containers or lists, what should be done about top-level YANG data nodes that are not containers or lists? Are they allowed in RESTCONF?

- Can a choice be a resource? YANG choices are invisible to RESTCONF at this time.

B.9. .well-known usage

- Does RESTCONF need to Use a .well-known link relation to to re-map API entry point?

The client first discovers the server's root for the RESTCONF API. In this example, it is "/api/restconf":

```
Request
-------
GET /.well-known/host-meta users HTTP/1.1
Host: example.com
Accept: application/xrd+xml

Response
--------
HTTP/1.1 200 OK
Content-Type: application/xrd+xml
Content-Length: nnn


<XRD xmlns='http://docs.oasis-open.org/ns/xri/xrd-1.0'>
    <Link rel='restconf' href='/api/restconf'/>
</XRD>
```

Once discovering the RESTCONF API root, the client MUST prepend it to any access to a RESTCONF resource:

```
Request
-------
GET /api/restconf/version  HTTP/1.1
Host: example.com
Accept: application/yang.api+json

Response
--------
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Last-Modified: Sun, 22 Apr 2012 01:00:14 GMT
Content-Type: application/yang.api+json

{ "version": "1.0" }
```

B.10. _self links for HATEOAS support

- Should there be a mode where the client can request that the resource identifier is returned in a GET request?

B.11. netconf-state monitoring support

- Should long-term RESTCONF operations (i.e. SSE long-poll) be considered sessions with regards to NETCONF monitoring "session" list? If so, what text is needed in RESTCONF draft to standardize the RESTCONF session entries?

B.12. secure transport

- Details to support secure operation over TLS are needed

- Security considerations need to be written

Appendix C. Example YANG Module

The example YANG module used in this document represents a simple media jukebox interface.

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
   |  +--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
```

C.1. example-jukebox YANG Module

```
module example-jukebox {

   namespace "http://example.com/ns/example-jukebox";
   prefix "jbox";
   import ietf-restconf { prefix rc; }
```

```
     organization "Example, Inc.";
     contact "support at example.com";
     description "Example Jukebox Data Model Module";
     revision "2013-12-21" {
       description "Initial version.";
       reference "example.com document 1-4673";
     }

     identity genre {
       description "Base for all genre types";
     }

     // abbreviated list of genre classifications
     identity alternative {
       base genre;
       description "Alternative music";
     }
     identity blues {
       base genre;
       description "Blues music";
     }
     identity country {
       base genre;
       description "Country music";
     }
     identity jazz {
       base genre;
       description "Jazz music";
     }
     identity pop {
       base genre;
       description "Pop music";
     }
     identity rock {
       base genre;
       description "Rock music";
     }

     container jukebox {
       presence
         "An empty container indicates that the jukebox
          service is available";

       description
         "Represents a jukebox resource, with a library, playlists,
          and a play operation.";

       container library {
```

```
          description "Represents the jukebox library resource.";

          list artist {
            key name;

            description
              "Represents one artist resource within the
               jukebox library resource.";

            leaf name {
              type string {
                length "1 .. max";
              }
              description "The name of the artist.";
            }

            list album {
              key name;

              description
                "Represents one album resource within one
                 artist resource, within the jukebox library.";

              leaf name {
                type string {
                  length "1 .. max";
                }
                description "The name of the album.";
              }

              leaf genre {
                type identityref { base genre; }
                description
                  "The genre identifying the type of music on
                   the album.";
              }

              leaf year {
                type uint16 {
                  range "1900 .. max";
                }
                description "The year the album was released";
              }

              list song {
                key name;

                description
```

```
                  "Represents one song resource within one
                   album resource, within the jukebox library.";

                leaf name {
                  type string {
                     length "1 .. max";
                  }
                  description "The name of the song";
                }
                leaf location {
                  type string;
                  mandatory true;
                  description
                   "The file location string of the
                    media file for the song";
                }
                leaf format {
                  type string;
                  description
                    "An identifier string for the media type
                     for the file associated with the
                     'location' leaf for this entry.";
                }
                leaf length {
                  type uint32;
                  units "seconds";
                  description
                    "The duration of this song in seconds.";
                }
              }   // end list 'song'
            }   // end list 'album'
          }  // end list 'artist'

          leaf artist-count {
             type uint32;
             units "songs";
             config false;
             description "Number of artists in the library";
          }
          leaf album-count {
             type uint32;
             units "albums";
             config false;
             description "Number of albums in the library";
          }
          leaf song-count {
             type uint32;
             units "songs";
```

```
             config false;
             description "Number of songs in the library";
           }
         }  // end library

         list playlist {
           key name;

           description
             "Example configuration data resource";

           leaf name {
             type string;
             description
               "The name of the playlist.";
           }
           leaf description {
             type string;
             description
               "A comment describing the playlist.";
           }
           list song {
             key index;
             ordered-by user;

             description
               "Example nested configuration data resource";

             leaf index {    // not really needed
               type uint32;
               description
                 "An arbitrary integer index for this
                  playlist song.";
             }
             leaf id {
               type rc:data-resource-identifier;
               mandatory true;
               description
                 "Song identifier. Must identify an instance of
                  /jukebox/library/artist/album/song/name.";
             }
           }
         }

         container player {
           description
             "Represents the jukebox player resource.";
```

```
          leaf gap {
            type decimal64 {
              fraction-digits 1;
              range "0.0 .. 2.0";
            }
            units "tenths of seconds";
            description "Time gap between each song";
          }
        }
      }

      rpc play {
        description "Control function for the jukebox player";
        input {
          leaf playlist {
            type string;
            mandatory true;
            description "playlist name";
          }
          leaf song-number {
            type uint32;
            mandatory true;
            description "Song number in playlist to play";
          }
        }
      }
   }
```

Appendix D. RESTCONF Message Examples

The examples within this document use the normative YANG module defined in Section 7 and the non-normative example YANG module defined in Appendix C.1.

This section shows some typical RESTCONF message exchanges.

D.1. Resource Retrieval Examples

D.1.1. Retrieve the Top-level API Resource

The client may start by retrieving the top-level API resource, using the entry point URI "/restconf".

```
GET /restconf   HTTP/1.1
Host: example.com
Accept: application/yang.api+json
```

The server might respond as follows:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Content-Type: application/yang.api+json
```


```
{
  "ietf-restconf:restconf": {
    "data" : [ null ],
    "modules": {
      "module": [
        {
          "name" : "example-jukebox",
          "revision" : "2013-12-21",
          "namespace" : "http://example.com/ns/example-jukebox",
          "schema" : [ null ]
        }
      ]
    },
    "operations" : {
       "play" : [ null ]
    },
    "streams" : {
      "stream" : [
        {
          "name" : "NETCONF",
          "description" : "default NETCONF event stream",
          "replay-support" : true,
          "replay-log-creation-time:" : "2007-07-08T00:00:00Z",
          "events" : [ null ]
        }
      ]
    },
    "version": "1.0"
  }
}
```


To request that the response content to be encoded in XML, the "Accept" header can be used, as in this example request:

```
GET /restconf HTTP/1.1
Host: example.com
Accept: application/yang.api+xml
```

The server will return the same response either way, which might be as follows :

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.api+xml
```


```
<restconf xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <data/>
  <modules>
    <module>
      <name>example-jukebox</name>
      <revision>2013-12-21</revision>
      <namespace>
        http://example.com/ns/example-jukebox
      </namespace>
      <schema />
    </module>
  </modules>
  <operations>
    <play xmlns="http://example.com/ns/example-jukebox"/>
  </operations>
  <streams>
    <stream>
      <name>NETCONF</name>
      <description>default NETCONF event stream
      </description>
      <replay-support>true</replay-support>
      <replay-log-creation-time>
         2007-07-08T00:00:00Z
      </replay-log-creation-time>
      <events/>
    </stream>
  </streams>
  <version>1.0</version>
</restconf>
```


D.1.2. Retrieve The Server Module Information

In this example the client is retrieving the modules resource from the server in JSON format:

```
GET /restconf/modules  HTTP/1.1
Host: example.com
Accept: application/yang.api+json
```

The server might respond as follows.

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Last-Modified: Sun, 22 Apr 2012 01:00:14 GMT
Content-Type: application/yang.api+json
```


```
{
  "ietf-restconf:modules": {
    "module": [
      {
        "name" : "foo",
        "revision" : "2012-01-02",
        "schema" : [null],
        "namespace" : "http://example.com/ns/foo",
        "feature" : [ "feature1", "feature2" ]
      },
      {
        "name" : "foo-types",
        "revision" : "2012-01-05",
        "schema" : [null],
        "namespace" : "http://example.com/ns/foo-types"
      },
      {
        "name" : "bar",
        "revision" : "2012-11-05",
        "schema" : [null],
        "namespace" : "http://example.com/ns/bar",
        "feature" : [ "bar-ext" ],
        "submodule" : [
          {
            "name" : "bar-submod1",
            "revision" : "2012-11-05",
            "schema" : [null]
          },
          {
            "name" : "bar-submod2",
            "revision" : "2012-11-05",
            "schema" : [null]
          }
        ]
      }
    ]
  }
}
```

D.2. Edit Resource Examples

D.2.1. Create New Data Resources

To create a new "artist" resource within the "library" resource, the client might send the following request.

```
POST /restconf/data/example-jukebox:jukebox/library HTTP/1.1
Host: example.com
Content-Type: application/yang.data+json

{ "example-jukebox:artist" : {
    "name" : "Foo Fighters"
  }
}
```

If the resource is created, the server might respond as follows. Note that the "Location" header line is wrapped for display purposes only:

```
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2012 17:02:00 GMT
Server: example-server
Location: http://example.com/restconf/data/
     example-jukebox:jukebox/library/artist/Foo%20Fighters
Last-Modified: Mon, 23 Apr 2012 17:02:00 GMT
ETag: b3830f23a4c
```

To create a new "album" resource for this artist within the "jukebox" resource, the client might send the following request. Note that the request URI header line is wrapped for display purposes only:

```
POST /restconf/data/example-jukebox:jukebox/
   library/artist/Foo%20Fighters  HTTP/1.1
Host: example.com
Content-Type: application/yang.data+json

{
  "example-jukebox:album" : {
    "name" : "Wasting Light",
    "genre" : "example-jukebox:alternative",
    "year" : 2012    # note this is the wrong date
  }
}
```

If the resource is created, the server might respond as follows. Note that the "Location" header line is wrapped for display purposes only:

```
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2012 17:03:00 GMT
Server: example-server
Location: http://example.com/restconf/data/
  example-jukebox:jukebox/library/artist/Foo%20Fighters/
  album/Wasting%20Light
Last-Modified: Mon, 23 Apr 2012 17:03:00 GMT
ETag: b8389233a4c
```

D.2.2. Detect Resource Entity Tag Change

In this example, the server just supports the mandatory datastore last-changed timestamp. The client has previously retrieved the "Last-Modified" header and has some value cached to provide in the following request to patch an "album" list entry with key value "Wasting Light". Only the "year" field is being updated.

```
PATCH /restconf/data/example-jukebox:jukebox/
  library/artist/Foo%20Fighters/album/Wasting%20Light/year
  HTTP/1.1
Host: example.com
Accept: application/yang.data+json
If-Unmodified-Since: Mon, 23 Apr 2012 17:01:00 GMT
Content-Type: application/yang.data+json

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

In this example the datastore resource has changed since the time specified in the "If-Unmodified-Since" header. The server might respond:

```
HTTP/1.1 412 Precondition Failed
Date: Mon, 23 Apr 2012 19:01:00 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 17:45:00 GMT
ETag: b34aed893a4c
```

D.3. Query String Parameter Examples

D.3.1. "content" Parameter

The "content" parameter is used to select the type of data sub-resources (configuration and/or not configuration) that are returned by the server for a GET method request.

In this example, a simple YANG list that has configuration and non-configuration sub-resources.

```
  container events
    list event {
      key name;
      leaf name { type string; }
      leaf description { type string; }
      leaf event-count {
        type uint32;
        config false;
      }
    }
  }
```

Example 1: content=all

To retrieve all the sub-resources, the "content" parameter is set to "all". The client might send:

```
GET /restconf/data/example-events:events?content=all
    HTTP/1.1
Host: example.com
Accept: application/yang.data+json
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json
```


```
{
  "example-events:events" : {
    "event" : [
      {
        "name" : "interface-up",
        "description" : "Interface up notification count",
        "event-count" : 42
      },
      {
        "name" : "interface-down",
        "description" : "Interface down notification count",
        "event-count" : 4
      }
    ]
  }
}
```

Example 2: content=config

To retrieve only the configuration sub-resources, the "content" parameter is set to "config" or omitted since this is the default value. Note that the "ETag" and "Last-Modified" headers are only returned if the content parameter value is "config".

```
GET /restconf/data/example-events:events?content=config
   HTTP/1.1
Host: example.com
Accept: application/yang.data+json
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:11:30 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
ETag: eeeada438af
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json

{
  "example-events:events" : {
    "event" : [
      {
        "name" : "interface-up",
        "description" : "Interface up notification count"
      },
      {
        "name" : "interface-down",
        "description" : "Interface down notification count"
      }
    ]
  }
}
```

Example 3: content=non-config

To retrieve only the non-configuration sub-resources, the "content" parameter is set to "non-config". Note that configuration ancestors (if any) and list key leafs (if any) are also returned. The client might send:

```
GET /restconf/data/example-events:events?content=non-config
   HTTP/1.1
Host: example.com
```

```
      Accept: application/yang.data+json
```

The server might respond:

```
      HTTP/1.1 200 OK
      Date: Mon, 23 Apr 2012 17:11:30 GMT
      Server: example-server
      Cache-Control: no-cache
      Pragma: no-cache
      Content-Type: application/yang.data+json
```


```
      {
        "example-events:events" : {
          "event" : [
            {
              "name" : "interface-up",
              "event-count" : 42
            },
            {
              "name" : "interface-down",
              "event-count" : 4
            }
          ]
        }
      }
```


D.3.2. "depth" Parameter

The "depth" parameter is used to limit the number of levels of sub-resources that are returned by the server for a GET method request.

This example shows how different values of the "depth" parameter would affect the reply content for retrieval of the top-level "jukebox" data resource.

Example 1: depth=unbounded

To retrieve all the sub-resources, the "depth" parameter is not present or set to the default value "unbounded". Note that some strings are wrapped for display purposes only.

```
      GET /restconf/data/example-jukebox:jukebox?depth=unbounded
         HTTP/1.1
      Host: example.com
      Accept: application/yang.data+json
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json
```


```
{
  "example-jukebox:jukebox" : {
    "library" : {
      "artist" : [
        {
          "name" : "Foo Fighters",
          "album" : [
            {
              "name" : "Wasting Light",
              "genre" : "example-jukebox:alternative",
              "year" : 2011,
              "song" : [
                {
                  "name" : "Wasting Light",
                  "location" :
                    "/media/foo/a7/wasting-light.mp3",
                  "format" : "MP3",
                  "length" " 286
                },
                {
                  "name" : "Rope",
                  "location" : "/media/foo/a7/rope.mp3",
                  "format" : "MP3",
                  "length" " 259
                }
              ]
            }
          ]
        }
      ]
    },
    "playlist" : [
      {
        "name" : "Foo-One",
        "description" : "example playlist 1",
        "song" : [
          {
            "index" : 1,
            "id" : "http://example.com/restconf/data/
                  example-jukebox:jukebox/library/artist/
                  Foo%20Fighters/album/Wasting%20Light/
```

```
                        song/Rope"
                },
                {
                  "index" : 2,
                  "id" : "http://example.com/restconf/data/
                        example-jukebox:jukebox/library/artist/
                        Foo%20Fighters/album/Wasting%20Light/song/
                        Bridge%20Burning"
                }
              ]
            }
          ],
          "player" : {
            "gap" : 0.5
          }
        }
      }
```


Example 2: depth=1

To determine if 1 or more resource instances exist for a given target resource, the value "1" is used.

```
GET /restconf/data/example-jukebox:jukebox?depth=1 HTTP/1.1
Host: example.com
Accept: application/yang.data+json
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json

{
  "example-jukebox:jukebox" : [null]
}
```

Example 3: depth=3

To limit the depth level to the target resource plus 2 sub-resource layers the value "3" is used.

```
GET /restconf/data/example-jukebox:jukebox?depth=3 HTTP/1.1
Host: example.com
Accept: application/yang.data+json
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json
```


```
{
  "example-jukebox:jukebox" : {
    "library" : {
      "artist" : [ null ]
    },
    "playlist" : [
      {
        "name" : "Foo-One",
        "description" : "example playlist 1",
        "song" : [ null ]
      }
    ],
    "player" : {
      "gap" : 0.5
    }
  }
}
```


D.3.3. "filter" Parameter

The following URIs show some examples of notification filter specifications (lines wrapped for display purposes only):

```
// filter = /event/eventClass='fault'
GET /restconf/streams/stream/NETCONF/events?
   filter=%2Fevent%2FeventClass%3D'fault'

// filter = /event/severityCode<=4
GET /restconf/streams/stream/NETCONF/events?
   filter=%2Fevent%2FseverityCode%3C%3D4

// filter = /linkUp|/linkDown
GET /restconf/streams/stream/SNMP/events?
   filter=%2FlinkUp%7C%2FlinkDown

// filter = /*/reportingEntity/card!='Ethernet0'
GET /restconf/streams/stream/NETCONF/events?
   filter=%2F*%2FreportingEntity%2Fcard%21%3D'Ethernet0'

// filter = /*/email-addr[contains(.,'company.com')]
GET /restconf/streams/stream/critical-syslog/events?
   filter=%2F*%2Femail-addr[contains(.%2C'company.com')]

// Note: the module name is used as prefix.
// filter = (/example-mod:event1/name='joe' and
//            /example-mod:event1/status='online')
GET /restconf/streams/stream/NETCONF/events?
  filter=(%2Fexample-mod%3Aevent1%2Fname%3D'joe'%20and
          %20%2Fexample-mod%3Aevent1%2Fstatus%3D'online')
```

D.3.4. "insert" Parameter

In this example, a new first entry in the "Foo-One" playlist is being created.

Request from client:

```
POST /restconf/data/example-jukebox:jukebox/
  playlist/Foo-One?insert=first HTTP/1.1
Host: example.com
Content-Type: application/yang.data+json

{
  "example-jukebox:song" : {
     "index" : 1,
     "id" : "/example-jukebox:jukebox/library/artist/
         Foo%20Fighters/album/Wasting%20Light/song/Rope"
   }
}
```

Response from server:

```
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2012 13:01:20 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
Location: http://example.com/restconf/data/
   example-jukebox:jukebox/playlist/Foo-One/song/1
ETag: eeeada438af
```

D.3.5. "point" Parameter

Example:

In this example, the client is inserting a new "song" resource within an "album" resource after another song. The request URI is split for display purposes only.

Request from client:

```
POST /restconf/data/example-jukebox:jukebox/
   library/artist/Foo%20Fighters/album/Wasting%20Light?
   insert=after&point=%2Fexample-jukebox%3Ajukebox%2F
   library%2Fartist%2FFoo%20Fighters%2Falbum%2F
   Wasting%20Light%2Fsong%2FBridge%20Burning   HTTP/1.1
Host: example.com
Content-Type: application/yang.data+json

{
  "example-jukebox:song" : {
     "name" : "Rope",
     "location" : "/media/foo/a7/rope.mp3",
     "format" : "MP3",
     "length" : 259
  }
}
```

Response from server:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 13:01:20 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
ETag: abcada438af
```

D.3.6. "select" Parameter

TBD

D.3.7. "start-time" Parameter

TBD

D.3.8. "stop-time" Parameter

TBD

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