

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 6, 2015

A. Bierman
YumaWorks
M. Bjorklund
Tail-f Systems
K. Watsen
Juniper Networks
June 4, 2015

RESTCONF Protocol
draft-ietf-netconf-restconf-05

Abstract

This document describes an HTTP-based protocol that provides a programmatic interface 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 December 6, 2015.

Copyright Notice

Copyright (c) 2015 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	4
1.1.	Simple Subset of NETCONF Functionality	5
1.2.	Data Model Driven API	6
1.3.	Coexistence with NETCONF	7
1.4.	Terminology	8
1.4.1.	NETCONF	8
1.4.2.	HTTP	8
1.4.3.	YANG	9
1.4.4.	Terms	10
1.4.5.	URI Template	11
1.4.6.	Tree Diagrams	11
2.	Transport Protocol Requirements	12
2.1.	Integrity and Confidentiality	12
2.2.	HTTPS with X.509v3 Certificates	12
2.3.	Certificate Validation	12
2.4.	Authenticated Server Identity	12
2.5.	Authenticated Client Identity	12
3.	Resources	13
3.1.	Root Resource Discovery	14
3.2.	RESTCONF Resource Types	15
3.3.	API Resource	15
3.3.1.	{+restconf}/data	16
3.3.2.	{+restconf}/operations	17
3.4.	Datastore Resource	17
3.4.1.	Edit Collision Detection	18
3.5.	Data Resource	18
3.5.1.	Encoding Data Resource Identifiers in the Request URI	19
3.5.2.	Defaults Handling	22
3.6.	Operation Resource	22
3.6.1.	Encoding Operation Input Parameters	23
3.6.2.	Encoding Operation Output Parameters	24
3.6.3.	Encoding Operation Errors	25
3.7.	Schema Resource	25
3.8.	Event Stream Resource	26
3.9.	Errors Media Type	27
4.	Operations	27
4.1.	OPTIONS	28
4.2.	HEAD	28
4.3.	GET	28
4.4.	POST	29
4.4.1.	Create Resource Mode	30
4.4.2.	Invoke Operation Mode	31
4.5.	PUT	31

4.6.	PATCH	32
4.6.1.	Plain Patch	33
4.7.	DELETE	34
4.8.	Query Parameters	34
4.8.1.	The "content" Query Parameter	35
4.8.2.	The "depth" Query Parameter	36
4.8.3.	The "fields" Query Parameter	36
4.8.4.	The "insert" Query Parameter	37
4.8.5.	The "point" Query Parameter	38
4.8.6.	The "filter" Query Parameter	38
4.8.7.	The "start-time" Query Parameter	39
4.8.8.	The "stop-time" Query Parameter	40
4.8.9.	The "with-defaults" Query Parameter	40
5.	Messages	41
5.1.	Request URI Structure	41
5.2.	Message Headers	43
5.3.	Message Encoding	44
5.4.	RESTCONF Meta-Data	44
5.5.	Return Status	44
5.6.	Message Caching	45
6.	Notifications	45
6.1.	Server Support	45
6.2.	Event Streams	45
6.3.	Subscribing to Receive Notifications	48
6.3.1.	NETCONF Event Stream	49
6.4.	Receiving Event Notifications	49
7.	Error Reporting	51
7.1.	Error Response Message	53
8.	RESTCONF module	55
9.	RESTCONF Monitoring	61
9.1.	restconf-state/capabilities	61
9.1.1.	Query Parameter URIs	62
9.1.2.	The "defaults" Protocol Capability URI	62
9.2.	restconf-state/streams	63
9.3.	RESTCONF Monitoring Module	63
10.	YANG Module Library	67
10.1.	modules	67
10.1.1.	modules/module	68
11.	IANA Considerations	68
11.1.	The "restconf" Relation Type	68
11.2.	YANG Module Registry	68
11.3.	application/yang Media Sub Types	69
11.4.	RESTCONF Capability URNs	70
12.	Security Considerations	71
13.	Acknowledgements	72
14.	References	72
14.1.	Normative References	72
14.2.	Informative References	75

Appendix A.	Change Log	75
A.1.	04 - 05	75
A.2.	03 - 04	76
A.3.	02 - 03	77
A.4.	01 - 02	77
A.5.	00 - 01	78
A.6.	bierman:restconf-04 to ietf:restconf-00	79
Appendix B.	Open Issues	79
Appendix C.	Example YANG Module	79
C.1.	example-jukebox YANG Module	80
Appendix D.	RESTCONF Message Examples	85
D.1.	Resource Retrieval Examples	86
D.1.1.	Retrieve the Top-level API Resource	86
D.1.2.	Retrieve The Server Module Information	87
D.1.3.	Retrieve The Server Capability Information	88
D.2.	Edit Resource Examples	89
D.2.1.	Create New Data Resources	89
D.2.2.	Detect Resource Entity Tag Change	90
D.2.3.	Edit a Datastore Resource	91
D.3.	Query Parameter Examples	92
D.3.1.	"content" Parameter	92
D.3.2.	"depth" Parameter	95
D.3.3.	"fields" Parameter	98
D.3.4.	"insert" Parameter	99
D.3.5.	"point" Parameter	100
D.3.6.	"filter" Parameter	101
D.3.7.	"start-time" Parameter	101
D.3.8.	"stop-time" Parameter	101
D.3.9.	"with-defaults" Parameter	102
Authors' Addresses	103

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 event notifications within a networking device, in a modular and extensible manner.

This document describes an HTTP [RFC7230] based protocol called RESTCONF, 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 event notifications. RESTCONF uses HTTP operations to provide CRUD operations on a NETCONF datastore containing YANG-

defined data. Since NETCONF protocol operations are not relevant, the user should not need any prior knowledge of NETCONF in order to use RESTCONF.

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 [[RFC7158](#)].

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

1.1. Simple Subset of NETCONF Functionality

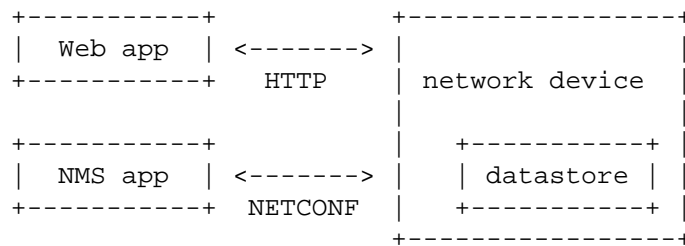
An HTTP-based management protocol does not need to mirror the functionality of the NETCONF protocol, but it needs to be compatible with NETCONF. A simplified transaction model is needed that allows basic CRUD operations on a hierarchy of conceptual resources. This represents a limited subset of the transaction capabilities of the NETCONF protocol.

The HTTP POST, PUT, PATCH, and DELETE methods are used to edit data resources represented by YANG data models. These basic edit operations allow the running configuration to be altered in an all-or-none fashion. This is similar to the "rollback-on-error" capability in NETCONF. Edits are usually applied to one data resource instance at a time.

The base RESTCONF protocol is intentionally simple to allow deployment for as many use cases as possible. Additional functionality can be defined in external documents, outside the scope of this document.

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

The following figure shows the system components:



1.2. Data Model Driven API

RESTCONF combines the simplicity of the HTTP protocol with the predictability and automation potential of a schema-driven API. Using YANG, a client can predict all resource endpoints, much like using URI Templates [RFC6570], but in a more holistic manner. This strategy obviates the need for responses provided by the server to contain HATEOAS links, originally described in Roy Fielding's doctoral dissertation [rest-dissertation].

In contrast, a REST client using HATEOAS principles would not use any data modeling language to define the application-specific content of the API. The client would need to discover each new child resource as it traverses the URIs to discover the server capabilities. This approach has the following 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 models 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 using the "ietf-yang-library" YANG module, defined in [\[I-D.ietf-netconf-yang-library\]](#). The server MUST implement the "ietf-yang-library" module, which SHOULD identify all the YANG modules used by the server.

The conceptual datastore contents, data-model-specific operations and event notifications are identified by this set of YANG modules. 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. Coexistence with NETCONF

RESTCONF can be implemented on a device that supports NETCONF.

If the device supports :writable-running, all edits to configuration nodes in {+restconf}/data are performed in the running configuration datastore.

Otherwise, if the device supports :candidate, all edits to configuration nodes in {+restconf}/data are performed in the candidate configuration datastore. The candidate is automatically committed to running after a successful edit.

If the device supports :startup, the device automatically copies the content of running to startup after running has been updated as a consequence of a RESTCONF edit operation.

If a datastore that would be modified by a RESTCONF operation has an active lock, the RESTCONF edit operation MUST fail with a 409 (Conflict) error code.

1.4. 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].

1.4.1. NETCONF

The following terms are defined in [RFC6241]:

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

1.4.2. HTTP

The following terms are defined in [RFC3986]:

- o fragment
- o path
- o query

The following terms are defined in [RFC7230]:

- o header
- o message-body
- o request-line
- o request URI
- o status-line

The following terms are defined in [[RFC7231](#)]:

- o method
- o request
- o resource

The following terms are defined in [[RFC7232](#)]:

- o entity tag

1.4.3. 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

1.4.4. 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".
- o data resource: a resource with the media type "application/yang.data+xml" or "application/yang.data+json". Containers, leafs, list entries and anyxml nodes can be data resources.
- o datastore resource: a resource with the media type "application/yang.datastore+xml" or "application/yang.datastore+json". Represents a datastore.
- o edit operation: a RESTCONF operation on a data resource using either a POST, PUT, PATCH, or DELETE method.
- o event stream resource: 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 an event notification stream. Also called a "stream resource".
- o media-type: HTTP uses Internet media types [[RFC2046](#)] in the Content-Type and Accept header fields in order to provide open and extensible data typing and type negotiation.
- o operation: the conceptual RESTCONF operation for a message, derived from the HTTP method, request URI, headers, and message-body.
- o operation resource: a resource with the media type "application/yang.operation+xml" or "application/yang.operation+json".
- o patch: a generic PATCH request on the target datastore or data resource. The media type of the message-body content will identify the patch type in use.
- o plain patch: a specific PATCH request type that can be used for simple merge operations.
- o query parameter: a parameter (and its value if any), encoded within the query component of the request URI.

- o RESTCONF capability: An optional RESTCONF protocol feature supported by the server, which is identified by an IANA registered NETCONF Capability URI, and advertised with an entry in the "capability" leaf-list in [Section 9.3](#).
- o retrieval request: a request using the GET or HEAD methods.
- o target resource: the resource that is associated with a particular message, identified by the "path" component of the request URI.
- o 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 stream list: the set of data resource instances that describe the event stream resources available from the server. This information is defined in the "ietf-restconf-monitoring" module as the "stream" list. It can be retrieved using the target resource "{+restconf}/data/ietf-restconf-monitoring:restconf-state/streams/stream". The stream list contains information about each stream, such as the URL to retrieve the event stream data.

1.4.5. URI Template

Throughout this document, the URI template [\[RFC6570\]](#) syntax "{+restconf}" is used to refer to the RESTCONF API entry point outside of an example. See [Section 3.1](#) for details.

For simplicity, all of the examples in this document assume "/restconf" as the discovered RESTCONF API root path.

1.4.6. Tree Diagrams

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

- o Brackets "[" and "]" enclose list keys.
- o Abbreviations before data node names: "rw" means configuration data (read-write) and "ro" state data (read-only).
- o Symbols after data node names: "?" means an optional node, "!" means a presence container, 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. Transport Protocol Requirements

2.1. Integrity and Confidentiality

HTTP [RFC7230] is an application layer protocol that may be layered on any reliable transport-layer protocol. RESTCONF is defined on top of HTTP, but due to the sensitive nature of the information conveyed, RESTCONF requires that the transport-layer protocol provides both data integrity and confidentiality, such as are provided by the TLS protocol [RFC5246].

2.2. HTTPS with X.509v3 Certificates

Given the nearly ubiquitous support for HTTP over TLS [RFC7230], RESTCONF implementations MUST support the "https" URI scheme, which has the IANA assigned default port 443. Consistent with the exclusive use of X.509v3 certificates for NETCONF over TLS [draft-ietf-netconf-rfc5539bis-10], use of certificates in RESTCONF is also limited to X.509v3 certificates.

2.3. Certificate Validation

When presented an X.509 certificate, the RESTCONF peer MUST use X.509 certificate path validation [RFC5280] to verify the integrity of the certificate. The presented X.509 certificate MAY also be considered valid if it matches a locally configured certificate fingerprint. If X.509 certificate path validation fails and the presented X.509 certificate does not match a locally configured certificate fingerprint, the connection MUST be terminated as defined in [RFC5246].

2.4. Authenticated Server Identity

The RESTCONF client MUST carefully examine the certificate presented by the RESTCONF server to determine if it meets the client's expectations. The RESTCONF client MUST check the identity of the server according to Section 6 of [RFC6125], including processing the outcome as described in Section 6.6 of [RFC6125].

2.5. Authenticated Client Identity

The RESTCONF server MUST authenticate client access to any protected resource using HTTP Authentication [RFC7235]. If the RESTCONF client is not authenticated to access a resource, the server MUST send a response with status code 401 (Unauthorized) and a WWW-Authenticate

header field containing at least one challenge applicable to the target resource. The RESTCONF server MAY advertise support for any number of authentication schemes but, in order to ensure interoperability, the RESTCONF server MUST advertise at least one of the following authentication schemes:

- o Basic [[draft-ietf-httpauth-basicauth-update-03](#)]
- o Digest [[draft-ietf-httpauth-digest-09](#)]
- o ClientCertificate [[draft-thomson-httpbis-cant-01](#)]

These authentication schemes are selected for to their similarity to the authentication schemes supported by NETCONF. In particular, the Basic and Digest authentication schemes both directly provide an identity and verification of a shared secret, much like NETCONF over SSH, when using the SSH "password" authentication method [[RFC4252](#)]. Similarly, the ClientCertificate authentication scheme is much like NETCONF over TLS's use of X.509 client-certificates. When using the ClientCertificate authentication scheme, the RESTCONF server MUST derive the identity of the RESTCONF client using the algorithm defined in Section 7 of [[draft-ietf-netconf-rfc5539bis-10](#)].

The RESTCONF client identity determined from any HTTP authentication scheme is hereafter known as the "RESTCONF username" and subject to the NETCONF Access Control Module (NACM) [[RFC6536](#)].

3. Resources

The RESTCONF protocol operates on a hierarchy of resources, starting with the top-level API resource itself ([Section 3.1](#)). 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 nested child 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 specific datastore contents, protocol operations, and event notifications. 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 data model specific operations, top-level data node resources, and event notification messages supported by the server.

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.

3.1. Root Resource Discovery

In line with the best practices defined by [RFC7320], RESTCONF enables deployments to specify where the RESTCONF API is located. When first connecting to a RESTCONF server, a RESTCONF client MUST determine the root of the RESTCONF API. The client discovers this by getting the `"/.well-known/host-meta"` resource ([RFC6415]) and using the `<Link>` element containing the `"restconf"` attribute :

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='/restconf'/>
</XRD>
```

Once discovering the RESTCONF API root, the client MUST prepend it to any subsequent request to a RESTCONF resource. For instance, using the `"/restconf"` path discovered above, the client can now determine the operations supported by the the server. In this example a custom `"play"` operation is supported:

Request

```
GET /restconf/operations 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
```

```
{ "operations" : { "play" : [ null ] } }
```

3.2. RESTCONF Resource Types

The RESTCONF protocol defines a set of 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+xml application/yang.api+json
Datastore	application/yang.datastore+xml application/yang.datastore+json
Data	application/yang.data+xml application/yang.data+json
Errors	application/yang.errors+xml application/yang.errors+json
Operation	application/yang.operation+xml application/yang.operation+json
Schema	application/yang

RESTCONF Media Types

3.3. API Resource

The API resource contains the entry points for the RESTCONF datastore and operation resources. It is the top-level resource located at `{+restconf}` and has the media type "application/yang.api+xml" or "application/yang.api+json".

YANG Tree Diagram for an API Resource:

```

+--rw restconf
  +--rw data
  +--rw operations

```

The "application/yang.api" restconf-media-type extension in the "ietf-restconf" module defined in [Section 8](#) is used to specify the structure and syntax of the conceptual child resources within the API resource.

The API resource can be retrieved with the GET method.

This resource has the following child resources:

Child Resource	Description
data	Contains all data resources
operations	Data-model specific operations

RESTCONF API Resource

3.3.1. {+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 3.4](#).

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 (see [Section 4.8.1](#)).

```

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

```

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+xml

<library xmlns="https://example.com/ns/example-jukebox">
  <artist-count>42</artist-count>
  <album-count>59</album-count>
  <song-count>374</song-count>
</library>
```

3.3.2. {+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 3.6](#).

3.4. Datastore Resource

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

This resource type is an abstraction of the system's underlying datastore implementation. It is used to simplify resource editing for the client. The RESTCONF datastore resource is a conceptual collection of all configuration and operational data that is present on the device.

Configuration edit transaction management and configuration persistence are handled by the server and not controlled by the client. A datastore resource can only be written directly with the PATCH method. Each RESTCONF edit of a datastore resource is saved to non-volatile storage in an implementation-specific matter by the server.

3.4.1. Edit Collision Detection

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

3.4.1.1. Timestamp

The last change time is maintained and the "Last-Modified" ([RFC7232], Section 2.2) header is 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 the top-level {+restconf}/data resource and SHOULD maintain last-modified timestamps for descendant resources. For all resources, the server MUST return the "Last-Modified" header when the resource is retrieved with the GET or HEAD methods. If the server does not maintain a timestamp for a resource, it MUST return the timestamp of the resource's ancestor, a process that may recurse up to the top-level {+restconf}/data resource. Only changes to configuration data resources within the datastore affect the timestamp.

3.4.1.2. Entity tag

A unique opaque string is maintained and the "ETag" ([RFC7232], Section 2.3) 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 an entity tag for the top-level {+restconf}/data resource and SHOULD maintain entity tags for descendant resources. For all resources, the server MUST return the "ETag" header when the resource is retrieved with the GET or HEAD methods. If the server does not maintain an entity tag for a resource, it MUST return the entity tag of the resource's ancestor, a process that may recurse up to the top-level {+restconf}/data resource. Only changes to configuration data resources within the datastore affect the entity tag.

3.5. Data Resource

A data resource represents a YANG data node that is a descendant node of a datastore resource. Each YANG-defined data node can be uniquely targeted by the request-line of an HTTP operation. Containers, leafs, list entries and anyxml nodes are data resources.

The representation maintained for each data resource is the YANG defined subtree for that node. HTTP operations on a data resource affect both the targeted data node and all its descendants, if any.

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 are 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 or all of the edit operations, depending on the target resource and the specific operation. Refer to [Section 4](#) 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.

3.5.1. Encoding Data Resource 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 path expressions are used instead.

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 3.5.1.1](#). 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) MUST be encoded according to the following rules:

- o The key leaf values for a data resource representing a YANG list MUST be encoded using one path segment [[RFC3986](#)].
- o If there is only one key leaf value, the path segment is constructed by having the list name followed by an "=" followed by the single key leaf value.
- o If there are multiple key leaf values, the value of each leaf identified in the "key" statement is encoded in the order specified in the YANG "key" statement, with a comma separating them.
- o The key value is specified as a string, using the canonical representation for the YANG data type. Any reserved characters MUST be encoded with escape sequences, according to [[RFC2396](#)], [Section 2.4](#).
- o All the components in the "key" statement MUST be encoded. Partial instance identifiers are not supported.
- o Quoted strings are supported in the key leaf values. Quoted strings MUST be used to express empty strings. (example: list=foo, '', baz).
- o The "list-instance" ABNF rule defined in [Section 3.5.1.1](#) represents the syntax of a list instance identifier.
- o 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:

```

container top {
  list list1 {
    key "key1 key2 key3";
    ...
    list list2 {
      key "key4 key5";
      ...
      leaf X { type string; }
    }
  }
}

```

For the above YANG definition, URI with key leaf values will be encoded as follows (line wrapped for display purposes only):

```

/restconf/data/example-top:top/list1=key1val,key2val,key3val3/
list2=key4val,key5val/X

```

3.5.1.1. ABNF For Data Resource Identifiers

The "api-path" ABNF syntax is used to construct RESTCONF path identifiers:

```

api-path = "/" |
  ("/" api-identifier
    0*("/" (api-identifier | list-instance )))

api-identifier = [module-name ":" ] identifier    ;; note 1

module-name = identifier

list-instance = api-identifier "=" key-value ["," key-value]*

key-value = string      ;; note 1

string = <a quoted or unquoted string>

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

```

Note 1: The syntax for "api-identifier" and "key-value" MUST conform to the JSON identifier encoding rules in Section 4 of [\[I-D.ietf-netmod-yang-json\]](#).

3.5.2. Defaults Handling

RESTCONF requires that a server report its default handling mode (see [Section 9.1.2](#) for details). If the optional "with-defaults" query parameter is supported by the server, a client may use it to control retrieval of default values (see [Section 4.8.9](#) for details).

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 child resources with default values, for the child resources that have not been given value yet, the server **MAY** return the default values that are in use by the server, in accordance with its reported default handling mode and query parameters passed by the client.

3.6. Operation Resource

An operation resource represents a protocol operation defined with the YANG "rpc" statement. It is invoked using a POST method on the operation resource.

```
POST {+restconf}/operations/<operation>
```

The <operation> field identifies the module name and rpc identifier string for the desired operation.

For example, if "module-A" 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
```

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 operation is successfully invoked, and if the "rpc" statement has an "output" section, then a message-body **MAY** be sent by the server in the response, otherwise the response message **MUST NOT** include a message-body in the response message, and **MUST** send a "204 No Content" status-line instead.

If the operation is not successfully invoked, then a message-body SHOULD be sent by the server, containing an "errors" resource, as defined in [Section 3.9](#).

3.6.1. 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.

```
module example-ops {
  namespace "https://example.com/ns/example-ops";
  prefix "ops";

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

  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:reboot HTTP/1.1
Host: example.com
Content-Type: application/yang.operation+xml
```

```
<input xmlns="https://example.com/ns/example-ops">
  <delay>600</delay>
  <message>Going down for system maintenance</message>
  <language>en-US</language>
</input>
```

The server might respond:

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

3.6.2. 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 "example-ops" YANG module defined in [Section 3.6.1](#) is used for the examples in this section.

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"
  }
}
```


3.6.3. Encoding Operation Errors

If any errors occur while attempting to invoke the operation, then an "errors" data structure is returned with the appropriate error status.

Using the "reset" operation example above, 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+xml

<input xmlns="https://example.com/ns/example-ops">
  <delay>-33</delay>
  <message>Going down for system maintenance</message>
  <language>en-US</language>
</input>
```

The server might respond with an "invalid-value" error:

```
HTTP/1.1 400 Bad Request
Date: Mon, 25 Apr 2012 11:10:30 GMT
Server: example-server
Content-Type: application/yang.errors+xml

<errors xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <error>
    <error-type>protocol</error-type>
    <error-tag>invalid-value</error-tag>
    <error-path xmlns:err="https://example.com/ns/example-ops">
      err:input/err:delay
    </error-path>
    <error-message>Invalid input parameter</error-message>
  </error>
</errors>
```

3.7. Schema Resource

The server can optionally support retrieval of the YANG modules it supports, using the "ietf-yang-library" module, defined in [\[I-D.ietf-netconf-yang-library\]](#).

To retrieve a YANG module, a client first needs to get the URL for retrieving the schema.

The client might send the following GET request message:

```
GET /restconf/data/ietf-yang-library:modules/module=
    example-jukebox,2014-07-03/schema HTTP/1.1
Host: example.com
Accept: application/yang.data+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.data+json
```

```
{
  "ietf-yang-library:schema":
    "https://example.com/mymodules/example-jukebox/2015-06-04"
}
```

Next the client needs to retrieve the actual YANG schema.

The client might send the following GET request message:

```
GET https://example.com/mymodules/example-jukebox/2015-06-04
    HTTP/1.1
Host: example.com
Accept: application/yang
```

The server might respond:

```
module example-jukebox {

    // contents of YANG module deleted for this example...

}
```

3.8. Event Stream Resource

An "event 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 6.3](#).

A notification stream functions according to the NETCONF Notifications specification [[RFC5277](#)]. The available streams can be retrieved from the stream list, which specifies the syntax and semantics of a stream resource.

3.9. Errors Media Type

An "errors" media type is a collection of error information that is sent as the message-body in a server response message, if an error occurs while processing a request message. It is not considered a resource type because no instances can be retrieved with a GET request.

The "ietf-restconf" YANG module contains the "application/yang.errors" restconf-media-type extension which specifies the syntax and semantics of an "errors" media type. RESTCONF error handling behavior is defined in [Section 7](#).

4. 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="create/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 when using a plain patch (see [Section 4.6.1](#)), other media-types may provide more granular control.

Access control mechanisms 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 [RFC7231]. This section defines the RESTCONF protocol usage for each HTTP method.

4.1. OPTIONS

The OPTIONS method is sent by the client to discover which methods are supported by the server for a specific resource (e.g., GET, POST, DELETE, etc.).

The server SHOULD implement this method, however the same information could be extracted from the YANG modules and the RESTCONF protocol specification.

If the PATCH method is supported, then the "Accept-Patch" header MUST be supported and returned in the response to the OPTIONS request, as defined in [RFC5789].

4.2. 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 message-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. 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 message-body is included.

4.3. 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.

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 the target resource, an error response containing a "403 Forbidden" or "404 Not Found" 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+xml
```

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+xml
Cache-Control: no-cache
Pragma: no-cache
ETag: a74eefc993a2b
Last-Modified: Mon, 23 Apr 2012 11:02:14 GMT

<album xmlns="http://example.com/ns/example-jukebox">
  <name>Wasting Light</name>
  <genre xmlns:g="http://example.com/ns/example-jukebox">
    g:alternative
  </genre>
  <year>2011</2011>
</album>
```

4.4. 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 child resource
Operation	Invoke a protocol operation

Resource Types that Support POST

4.4.1. 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 top-level resource or child resource, respectively. The message-body is expected to contain the content of a child resource to create within the parent (target resource). The data-model for the child tree is the subtree is defined by YANG for the child 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 8](#).

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" or "404 Not Found" status-line is returned to the client. All other error responses are handled according to the procedures defined in [Section 7](#).

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. Note that the "Location" header line is wrapped for display purposes only:

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

Refer to [Appendix D.2.1](#) for more resource creation examples.

4.4.2. 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 3.6](#) for details on operation resources.

If the POST request 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" or "404 Not Found" status-line is returned to the client. All other error responses are handled according to the procedures defined in [Section 7](#).

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
```

4.5. PUT

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

The only target resource media type that supports PUT is the data resource. The message-body is expected to contain the content used to create or replace the target resource.

The "insert" ([Section 4.8.4](#)) and "point" ([Section 4.8.5](#)) query parameters are supported by the PUT method for data resources.

Consistent with [\[RFC7231\]](#), if the PUT request 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" or "404 Not Found" status-line is returned to the client. All other error responses are handled according to the procedures defined in [Section 7](#).

Example:

An "album" child 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-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
```

[4.6. PATCH](#)

RESTCONF 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 media types supported by a server can be discovered by the client by sending an OPTIONS request (see [Section 4.1](#)).

If the target resource instance does not exist, the server MUST NOT create it.

If the PATCH request 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" or "404 Not Found" status-line is returned to the client. All other error responses are handled according to the procedures defined in [Section 7](#).

4.6.1. Plain Patch

The plain patch mechanism merges the contents of the message body with the target resource. If the target resource is a datastore resource (see [Section 3.4](#)), the message body MUST be either application/yang.datastore+xml or application/yang.datastore+json. If then the target resource is a data resource (see [Section 3.5](#)), then the message body MUST be either application/yang.data+xml or application/yang.data+json.

Plain patch can be used to create or update, but not delete, a child resource within the target resource. Please see [\[I-D.ietf-netconf-yang-patch\]](#) for an alternate media-type supporting more granular control.

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-line is wrapped for display purposes only:

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

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

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
```

4.7. DELETE

The DELETE method is used to delete the target resource. If the DELETE request 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" or "404 Not Found" status-line is returned to the client. All other error responses are handled according to the procedures defined in [Section 7](#).

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
```

4.8. 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
fields	GET	Request a subset of the target resource contents
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
start-time	GET	Replay buffer start time for event stream resources
stop-time	GET	Replay buffer stop time for event stream resources
with-defaults	GET	Control retrieval of default values

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.

Refer to [Appendix D.3](#) for examples of query parameter usage.

If vendors define additional query parameters, they SHOULD use a prefix (such as the enterprise or organization name) for query parameter names in order to avoid collisions with other parameters.

4.8.1. The "content" Query Parameter

The "content" parameter controls how descendant nodes of the requested data nodes will be processed in the reply.

The allowed values are:

Value	Description
config	Return only configuration descendant data nodes
nonconfig	Return only non-configuration descendant data nodes
all	Return all descendant data nodes

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" statement value of the requested data nodes. If the "config" value is "false", then the default for the "content" parameter is "nonconfig". If "config" is "true" then the default for the "content" parameter is "config".

This query parameter MUST be supported by the server.

4.8.2. The "depth" Query Parameter

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.

The value of the "depth" parameter is either an integer between 1 and 65535, or the string "unbounded". "unbounded" is the default.

This parameter is only allowed for GET methods on API, datastore, and data resources. A 400 Bad Request error is returned if it 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.

If the "depth" query parameter URI is listed in the "capability" leaf-list in [Section 9.3](#), then the server supports the "depth" query parameter.

4.8.3. The "fields" Query Parameter

The "fields" query parameter is used to optionally identify data nodes within the target resource to be retrieved in a GET method. The client can use this parameter to retrieve a subset of all nodes in a resource.

A value of the "fields" query parameter matches the following rule:

```
fields-expr = path '(' fields-expr / '*' ')' /  
              path ';' fields-expr /  
              path  
path = api-identifier [ '/' path ]
```

"api-identifier" is defined in [Section 3.5.1.1](#).

";" is used to select multiple nodes. For example, to retrieve only the "genre" and "year" of an album, use: "fields=genre;year".

Parentheses are used to specify sub-selectors of a node. For example, to retrieve only the "label" and "catalogue-number" of an album, use: "fields=admin(label;catalogue-number)".

"/" is used in a path to retrieve a child node of a node. For example, to retrieve only the "label" of an album, use: "fields=admin/label".

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 the "fields" query parameter URI is listed in the "capability" leaf-list in [Section 9.3](#), then the server supports the "fields" parameter.

4.8.4. The "insert" Query Parameter

The "insert" parameter is used to specify how a resource should be inserted within a user-ordered list.

The allowed values are:

Value	Description
first	Insert the new data as the new first entry.
last	Insert the new data as the new last entry.
before	Insert the new data before the insertion point, as specified by the value of the "point" parameter.
after	Insert the new data after the insertion point, as specified by the value of the "point" parameter.

The default value is "last".

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.

The "insert" query parameter MUST be supported by the server.

4.8.5. The "point" Query Parameter

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.

The value of the "point" parameter is a string that identifies the path to the insertion point object. The format is the same as a target resource URI string.

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.

The "point" query parameter MUST be supported by the server.

4.8.6. The "filter" Query Parameter

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 1.0 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 1.0.
- 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 event notification is delivered to the client.

If the "filter" query parameter URI is listed in the "capability" leaf-list in [Section 9.3](#), then the server supports the "filter" query parameter.

4.8.7. The "start-time" Query Parameter

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 stream list entry for the stream resource, then the server MUST return the HTTP error code 400 Bad Request.

The value of the "start-time" parameter is of type "date-and-time", defined in the "ietf-yang" YANG module [\[RFC6991\]](#).

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 being 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.

If this query parameter is supported by the server, then the "replay" query parameter URI MUST be listed in the "capability" leaf-list in [Section 9.3](#). The "stop-time" query parameter MUST also be supported by the server.

If the "replay-support" leaf is present in the "stream" entry (defined in [Section 9.3](#)) then the server MUST support the "start-time" and "stop-time" query parameters for that stream.

4.8.8. The "stop-time" Query Parameter

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.

The value of the "stop-time" parameter is of type "date-and-time", defined in the "ietf-yang" YANG module [RFC6991].

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.

If this query parameter is supported by the server, then the "replay" query parameter URI **MUST** be listed in the "capability" leaf-list in [Section 9.3](#). The "start-time" query parameter **MUST** also be supported by the server.

If the "replay-support" leaf is present in the "stream" entry (defined in [Section 9.3](#)) then the server **MUST** support the "start-time" and "stop-time" query parameters for that stream.

4.8.9. The "with-defaults" Query Parameter

The "with-defaults" parameter is used to specify how information about default data nodes should be returned in response to GET requests on data resources.

If the server supports this capability, then it **MUST** implement the behavior in [Section 4.5.1 of \[RFC6243\]](#), except applied to the RESTCONF GET operation, instead of the NETCONF operations.

Value	Description
report-all	All data nodes are reported
trim	Data nodes set to the YANG default are not reported
explicit	Data nodes set by the client are not reported
report-all-tagged	All data nodes are reported and defaults are tagged

If the "with-defaults" parameter is set to "report-all" then the server MUST adhere to the defaults reporting behavior defined in [Section 3.1 of \[RFC6243\]](#).

If the "with-defaults" parameter is set to "trim" then the server MUST adhere to the defaults reporting behavior defined in [Section 3.2 of \[RFC6243\]](#).

If the "with-defaults" parameter is set to "explicit" then the server MUST adhere to the defaults reporting behavior defined in [Section 3.3 of \[RFC6243\]](#).

If the "with-defaults" parameter is set to "report-all-tagged" then the server MUST adhere to the defaults reporting behavior defined in [Section 3.4 of \[RFC6243\]](#).

If the "with-defaults" parameter is not present then the server MUST adhere to the defaults reporting behavior defined in its "basic-mode" parameter for the "defaults" protocol capability URI, defined in [Section 9.1.2](#).

If the server includes the "with-defaults" query parameter URI in the "capability" leaf-list in [Section 9.3](#), then the "with-defaults" query parameter MUST be supported.

5. Messages

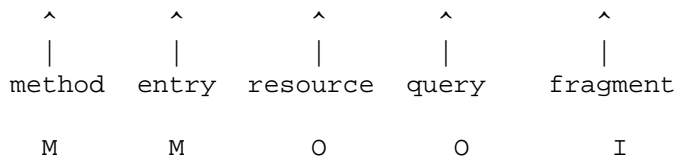
The RESTCONF protocol uses HTTP entities for messages. A single HTTP message corresponds to a single protocol method. Most messages can perform a single task on a single resource, such as retrieving a resource or editing a resource. The exception is the PATCH method, which allows multiple datastore edits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

5.1. 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:

<OP> /<restconf>/<path>?<query>#<fragment>



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 4](#).
- o entry: the root of the RESTCONF API configured on this HTTP server, discovered by getting the ".well-known/host-meta" resource, as described in [Section 3.1](#).
- 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. All query parameters are optional to implement by the server and optional to use by the client. Each query parameter is identified by a URI. The server MUST list the query parameter URIs it supports in the "capabilities" list defined in [Section 9.3](#).

There is a specific set of parameters defined, although the server MAY choose to support query 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.

5.2. 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 4](#) 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 message-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

5.3. Message Encoding

RESTCONF messages are encoded in HTTP according to [\[RFC7230\]](#). 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. A server MUST support XML encoding and MAY support JSON encoding. 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 [\[I-D.ietf-netmod-yang-json\]](#). 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.

5.4. 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. JSON encoding of meta-data is defined in [\[I-D.lhotka-netmod-yang-metadata\]](#).

5.5. 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 7.1](#).

5.6. 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.

6. Notifications

The RESTCONF protocol supports YANG-defined event notifications. The solution preserves aspects of NETCONF Event Notifications [[RFC5277](#)] while utilizing the Server-Sent Events [[W3C.CR-eventsource-20121211](#)] transport strategy.

6.1. 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 stream list. The server does not support RESTCONF notifications if an HTTP error code is returned (e.g., 404 Not Found).

6.2. 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 stream list.

The "restconf-state/streams" container definition in the "ietf-restconf-monitoring" module (defined in [Section 9.3](#)) is used to specify the structure and syntax of the conceptual child resources within the "streams" resource.

For example:

The client might send the following request:

```
GET /restconf/data/ietf-restconf-monitoring:restconf-state/  
streams HTTP/1.1  
Host: example.com  
Accept: application/yang.data+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-monitoring">
  <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>
    <access>
      <encoding>xml</encoding>
      <location>https://example.com/streams/NETCONF
      </location>
    </access>
    <access>
      <encoding>json</encoding>
      <location>https://example.com/streams/NETCONF-JSON
      </location>
    </access>
  </stream>
  <stream>
    <name>SNMP</name>
    <description>SNMP notifications</description>
    <replay-support>false</replay-support>
    <access>
      <encoding>xml</encoding>
      <location>https://example.com/streams/SNMP</location>
    </access>
  </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>
    <access>
      <encoding>xml</encoding>
      <location>
        https://example.com/streams/syslog-critical
      </location>
    </access>
  </stream>
</streams>
```

6.3. Subscribing to Receive Notifications

RESTCONF clients can determine the URL for the subscription resource (to receive notifications) by sending an HTTP GET request for the "location" leaf with the stream list entry. The value returned by the server can be used for the actual notification subscription.

The client will send an HTTP GET request for the URL returned by the server with the "Accept" type "text/event-stream".

The server will treat the connection as an event stream, using the Server Sent Events [[W3C.CR-eventsourcing-20121211](#)] 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:

The client might send the following request:

```
GET /restconf/data/ietf-restconf-monitoring:restconf-state/  
    streams/stream=NETCONF/encoding=xml/location HTTP/1.1  
Host: example.com  
Accept: application/yang.data+xml
```

The server might send the following response:

```
HTTP/1.1 200 OK  
Content-Type: application/yang.api+xml  
  
<location  
  xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring">  
  https://example.com/streams/NETCONF  
</location>
```

The RESTCONF client can then use this URL value to start monitoring the event stream:

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

```

6.3.1. 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. These query parameters are optional to implement, and only available if the server supports them.

Name	Section	Description
start-time	4.8.7	replay event start time
stop-time	4.8.8	replay event stop time
filter	4.8.6	boolean content filter

NETCONF Stream Query Parameters

The semantics and syntax for these query parameters are defined in the sections listed above. The YANG encoding MUST be converted to URL-encoded string for use in the request URI.

Refer to [Appendix D.3.6](#) for filter parameter examples.

6.4. 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 schema for the "notification" element in [Section 4 of \[RFC5277\]](#), except the XML namespace for this element is defined as:

```
urn:ietf:params:xml:ns:yang:ietf-restconf
```

For JSON encoding purposes, the module name for the "notification" element is "ietf-restconf".

Two child nodes within the "notification" container are expected, representing the event time and the event payload. The "event-time" node is defined within the "ietf-restconf" module namespace. The name and namespace of the payload element are determined by the YANG module containing the notification-stmt.

In the following example, the YANG module "example-mod" is used:

```
module example-mod {
  namespace "http://example.com/event/1.0";

  notification event {
    leaf event-class { type string; }
    container reporting-entity {
      leaf card { type string; }
    }
    leaf severity { type string; }
  }
}
```

An example SSE event 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:     <event-class>fault</event-class>
data:     <reporting-entity>
data:       <card>Ethernet0</card>
data:     </reporting-entity>
data:     <severity>major</severity>
data:   </event>
data: </notification>
```

An example SSE event notification encoded using JSON:

```
data: {
data:   "ietf-restconf:notification": {
data:     "event-time": "2013-12-21T00:01:00Z",
data:     "example-mod:event": {
data:       "event-class": "fault",
data:       "reporting-entity": { "card": "Ethernet0" },
data:       "severity": "major"
data:     }
data:   }
data: }
```

Alternatively, since neither XML nor JSON are whitespace sensitive, the above messages can be encoded onto a single line. For example:

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

XML:

```
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"><event-class>fault</event-class><re\
portingEntity><card>Ethernet0</card></reporting-entity><severity>\
major</severity></event></notification>
```

JSON:

```
data: {"ietf-restconf:notification":{"event-time":"2013-12-21\
T00:01:00Z","example-mod:event":{"event-class": "fault","repor\
tingEntity":{"card":"Ethernet0"},"severity":"major"}}}
```

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.

7. 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 message-body
201 Created	POST to create a resource success
202 Accepted	POST to create a resource accepted
204 No Content	Success without response message-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

7.1. 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 Forbidden"), then the server SHOULD send a response message-body containing the information described by the "errors" container definition within the YANG module [Section 8](#). The Content-Type of this response message MUST be application/yang.errors (see example below).

The client MAY specify the desired encoding for error messages by specifying the appropriate media-type in the Accept header. If no error media is specified, the server MUST assume that "application/yang.errors+xml" was specified. All of the examples in this document, except for the one below, assume the default XML encoding will be returned if there is an error.

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-path?     instance-identifier
    +--ro error-message?  string
    +--ro error-info

```

The semantics and syntax for RESTCONF error messages are defined in the "application/yang.errors" restconf-media-type extension in [Section 8](#).

Examples:

The following example shows an error returned for an "lock-denied" error that can occur if a NETCONF client has locked a datastore. The RESTCONF client is attempting to delete a data resource. Note that an Accept header is used to specify the desired encoding for the error message. This example's use of the Accept header is especially notable since the DELETE method typically doesn't return a message-body and hence Accept headers are typically not passed.

```

DELETE /restconf/data/example-jukebox:jukebox/
       library/artist=Foo%20Fighters/album=Wasting%20Light HTTP/1.1
Host: example.com
Accept: application/yang.errors+json

```

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.errors+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.errors+json

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

8. RESTCONF module

The "ietf-restconf" module defines conceptual definitions within an extension and two groupings, which are not meant to be implemented as datastore contents by a server. E.g., the "restconf" container is not intended to be implemented as a top-level data node (under the "/restconf/data" entry point).

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

<CODE BEGINS> file "ietf-restconf@2015-06-04.yang"

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

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

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

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

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

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

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

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

description

"This module contains conceptual YANG specifications for basic RESTCONF media type definitions used in RESTCONF protocol messages.

Note that the YANG definitions within this module do not represent configuration data of any kind.

The 'restconf-media-type' YANG extension statement provides a normative syntax for XML and JSON message encoding purposes.

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

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

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

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

// RFC Ed.: remove this note
// Note: extracted from draft-ietf-netconf-restconf-05.txt

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
revision 2015-06-04 {
```



```
description
  "Initial revision.";
reference
  "RFC XXXX: RESTCONF Protocol.";
}

extension restconf-media-type {
  argument media-type-id {
    yin-element true;
  }
}
// RFC Ed.: replace draft-ietf-netmod-yang-json with RFC number
// in the description below, and remove this note.
description
  "This extension is used to specify a YANG data structure which
  represents a conceptual RESTCONF media type.
  Data definition statements within this extension specify
  the generic syntax for the specific media type.

  YANG is mapped to specific encoding formats outside the
  scope of this extension statement. RFC 6020 defines XML
  encoding rules for all RESTCONF media types that use
  the '+xml' suffix. draft-ietf-netmod-yang-json defines
  JSON encoding rules for all RESTCONF media types that
  use the '+json' suffix.

  The 'media-type-id' parameter value identifies the media type
  that is being defined. It contains the string associated
  with the generic media type, i.e., no suffix is specified.

  This extension is ignored unless it appears as a top-level
  statement. It SHOULD contain data definition statements
  that result in exactly one container data node definition.
  This allows compliant translation to an XML instance
  document for each media type.

  The module name and namespace value for the YANG module using
  the extension statement is assigned to instance document data
  conforming to the data definition statements within
  this extension.

  The sub-statements of this extension MUST follow the
  'data-def-stmt' rule in the YANG ABNF.

  The XPath document root is the extension statement itself,
  such that the child nodes of the document root are
  represented by the data-def-stmt sub-statements within
  this extension. This conceptual document is the context
  for the following YANG statements:
```

- must-stmt
- when-stmt
- path-stmt
- min-elements-stmt
- max-elements-stmt
- mandatory-stmt
- unique-stmt
- ordered-by
- instance-identifier data type

The following data-def-stmt sub-statements have special meaning when used within a restconf-resource extension statement.

- The list-stmt is not required to have a key-stmt defined.
- The if-feature-stmt is ignored if present.
- The config-stmt is ignored if present.
- The available identity values for any 'identityref' leaf or leaf-list nodes is limited to the module containing this extension statement, and the modules imported into that module.

```
" ;
}

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

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

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 {
    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.";
}

leaf error-path {
  type instance-identifier;
  description
    "The YANG instance identifier associated
    with the error node.";
}

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

  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 the ietf-restconf-monitoring
        module.";
    }

    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;
        }
      ";
    }
  }
}
```

```

    }
  }
}

<CODE ENDS>

```

9. RESTCONF Monitoring

The "ietf-restconf-monitoring" module provides information about the RESTCONF protocol capabilities and event notification streams available from the server. Implementation is mandatory for RESTCONF servers, if any protocol capabilities or event notification streams are supported.

YANG Tree Diagram for "ietf-restconf-monitoring" module:

```

+--ro restconf-state
  +--ro capabilities
  |   +--ro capability*   inet:uri
  +--ro streams
    +--ro stream* [name]
      +--ro name           string
      +--ro description?   string
      +--ro replay-support? boolean
      +--ro replay-log-creation-time? yang:date-and-time
      +--ro access* [type]
        +--ro encoding   string
        +--ro location   inet:uri

```

9.1. restconf-state/capabilities

This mandatory container holds the RESTCONF protocol capability URIs supported by the server.

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

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

The server **MUST** include a "capability" URI leaf-list entry for the "defaults" mode used by the server, defined in [Section 9.1.2](#).

The server **MUST** include a "capability" URI leaf-list entry identifying each supported optional protocol feature. This includes

optional query parameters and MAY include other capability URIs defined outside this document.

9.1.1. Query Parameter URIs

A new set of RESTCONF Capability URIs are defined to identify the specific query parameters (defined in [Section 4.8](#)) supported by the server.

The server MUST include a "capability" leaf-list entry for each optional query parameter that it supports.

Name	Section	URI
depth	4.8.2	urn:ietf:params:restconf:capability:depth:1.0
fields	4.8.3	urn:ietf:params:restconf:capability:fields:1.0
filter	4.8.6	urn:ietf:params:restconf:capability:filter:1.0
replay	4.8.7	urn:ietf:params:restconf:capability:replay:1.0
with-defaults	4.8.9	urn:ietf:params:restconf:capability:with-defaults:1.0

RESTCONF Query Parameter URIs

9.1.2. The "defaults" Protocol Capability URI

This URI identifies the defaults handling mode that is used by the server for processing default leaves in requests for data resources. A parameter named "basic-mode" is required for this capability URI. The "basic-mode" definitions are specified in the "With-Defaults Capability for NETCONF" [[RFC6243](#)].

Name	URI
defaults	urn:ietf:params:restconf:capability:defaults:1.0

RESTCONF defaults capability URI

This protocol capability URI MUST be supported by the server, and the MUST be listed in the "capability" leaf-list in [Section 9.3](#).

Value	Description
report-all	No data nodes are considered default
trim	Values set to the YANG default-stmt value are default
explicit	Values set by the client are never considered default

If the "basic-mode" is set to "report-all" then the server MUST adhere to the defaults handling behavior defined in [Section 2.1 of \[RFC6243\]](#).

If the "basic-mode" is set to "trim" then the server MUST adhere to the defaults handling behavior defined in [Section 2.2 of \[RFC6243\]](#).

If the "basic-mode" is set to "explicit" then the server MUST adhere to the defaults handling behavior defined in [Section 2.3 of \[RFC6243\]](#).

Example: (split for display purposes only)

```
urn:ietf:params:restconf:capability:defaults:1.0?
  basic-mode=explicit
```

9.2. restconf-state/streams

This optional container provides access to the event notification streams supported by the server. The server MAY omit this container if no event notification streams are supported.

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 contains a URI that represents an event stream resource.

Stream resources are defined in [Section 3.8](#). Notifications are defined in [Section 6](#).

9.3. RESTCONF Monitoring Module

The "ietf-restconf-monitoring" module defines monitoring information for the RESTCONF protocol.

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-monitoring@2015-06-04.yang"

module ietf-restconf-monitoring {
  namespace "urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring";
  prefix "rcmon";

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

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

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

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

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

  description
    "This module contains monitoring information for the
    RESTCONF protocol.

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

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



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

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

// RFC Ed.: remove this note
// Note: extracted from draft-ietf-netconf-restconf-05.txt

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
revision 2015-06-04 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: RESTCONF Protocol.";
}

container restconf-state {
  config false;
  description
    "Contains RESTCONF protocol monitoring information.";

  container capabilities {
    description
      "Contains a list of protocol capability URIs";

    leaf-list capability {
      type inet:uri;
      description "A RESTCONF protocol capability URI.";
    }
  }

  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.
        If 'true', then the server MUST support the 'start-time'
        and 'stop-time' query parameters for this stream.";
    reference
        "RFC 5277, Section 3.4, <replaySupport> element.";
}

leaf replay-log-creation-time {
    when "../replay-support" {
        description
            "Only present if notification replay is supported";
    }
    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.";
}

list access {
    key type;
    min-elements 1;
    description
        "The server will create an entry in this list for each
        encoding format that is supported for this stream.
        The media type 'application/yang.stream' is expected
        for all event streams. This list identifies the
        sub-types supported for this stream.";

    leaf encoding {
        type string;
        description
```

```
        "This is the secondary encoding format within the
        'text/event-stream' encoding used by all streams.
        The type 'xml' is supported for the media type
        'application/yang.stream+xml'. The type 'json'
        is supported for the media type
        'application/yang.stream+json'.";
    }

    leaf location {
        type inet:uri;
        mandatory true;
        description
            "Contains a URL that represents the entry point
            for establishing notification delivery via server
            sent events.";
    }
}
}
}
}
}

<CODE ENDS>
```

10. YANG Module Library

The "ietf-yang-library" module defined in [I-D.ietf-netconf-yang-library] provides information about the YANG modules and submodules used by the RESTCONF server. Implementation is mandatory for RESTCONF servers. All YANG modules and submodules used by the server MUST be identified in the YANG module library.

10.1. modules

This mandatory container holds the identifiers for the YANG data model modules supported by the server.

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

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

10.1.1.1. modules/module

This mandatory list contains one entry for each YANG data model module supported by the server. There MUST be an instance of this list for every YANG module that is used by the server.

The contents of this list are defined in the "module" YANG list statement in [I-D.ietf-netconf-yang-library].

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

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

11. IANA Considerations

11.1. The "restconf" Relation Type

This specification registers the "restconf" relation type in the Link Relation Type Registry defined by [RFC5988]:

Relation Name: restconf

Description: Identifies the root of RESTCONF API as configured on this HTTP server. The "restconf" relation defines the root of the API defined in RFCXXXX. Subsequent revisions of RESTCONF will use alternate relation values to support protocol versioning.

Reference: RFC XXXX

11.2. YANG Module Registry

This document registers two URIs 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.

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

This document registers two YANG modules 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

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

11.3. 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
- errors
- operation
- stream

Type name: application

Subtype name: yang.xxx

Required parameters: none

Optional parameters: See [section 4.8](#) in RFC XXXX

Encoding considerations: 8-bit

Security considerations: See [Section 12](#) in RFC XXXX

Interoperability considerations: none

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

Published specification: RFC XXXX

11.4. RESTCONF Capability URNs

[Note to RFC Editor:

The RESTCONF Protocol Capability Registry does not yet exist;
Need to ask IANA to create it; remove this note for publication

]

This document defines a registry for RESTCONF capability identifiers. The name of the registry is "RESTCONF Capability URNs". The registry shall record for each entry:

- o the name of the RESTCONF capability. By convention, this name is prefixed with the colon ':' character.
- o the URN for the RESTCONF capability.

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

Index

Capability Identifier

:defaults

urn:ietf:params:restconf:capability:defaults:1.0

:depth

urn:ietf:params:restconf:capability:depth:1.0

:fields

urn:ietf:params:restconf:capability:fields:1.0

:filter

urn:ietf:params:restconf:capability:filter:1.0

:replay

urn:ietf:params:restconf:capability:replay:1.0

:with-defaults

urn:ietf:params:restconf:capability:with-defaults:1.0

12. Security Considerations

This section provides security considerations for the resources defined by the RESTCONF protocol. Security considerations for HTTPS are defined in [\[RFC2818\]](#). Security considerations for the content manipulated by RESTCONF can be found in the documents defining data models.

This document does not specify an authentication scheme, but it does require that an authenticated NETCONF username be associated with each HTTP request. The authentication scheme MAY be implemented in the underlying transport layer (e.g., client certificates) or within the HTTP layer (e.g., Basic Auth, OAuth, etc.). RESTCONF does not itself define an authentication mechanism. Authentication MUST occur in a lower layer. Implementors SHOULD provide a comprehensive authorization scheme with RESTCONF and ensure that the resulting NETCONF username is made available to the RESTCONF server.

Authorization of individual user access to operations and data MAY be configured via NETCONF Access Control Model (NACM) [\[RFC6536\]](#), as specified in [Section 4](#).

Configuration information is by its very nature sensitive. Its transmission in the clear and without integrity checking leaves devices open to classic eavesdropping and false data injection attacks. Configuration information often contains passwords, user

names, service descriptions, and topological information, all of which are sensitive. Because of this, this protocol SHOULD be implemented carefully with adequate attention to all manner of attack one might expect to experience with other management interfaces.

Different environments may well allow different rights prior to and then after authentication. When an operation is not properly authorized, the RESTCONF server MUST return HTTP error status code 401 Unauthorized. Note that authorization information can be exchanged in the form of configuration information, which is all the more reason to ensure the security of the connection.

13. Acknowledgements

The authors would like to thank the following people for their contributions to this document: Ladislav Lhotka, Juergen Schoenwaelder, Rex Fernando, Robert Wilton, and Jonathan Hansford.

14. References

14.1. Normative References

- [I-D.ietf-netconf-yang-library]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", [draft-ietf-netconf-yang-library-00](#) (work in progress), January 2015.
- [I-D.ietf-netmod-yang-json]
Lhotka, L., "JSON Encoding of Data Modeled with YANG", [draft-ietf-netmod-yang-json-02](#) (work in progress), November 2014.
- [I-D.lhotka-netmod-yang-metadata]
Lhotka, L., "Defining and Using Metadata with YANG", [draft-lhotka-netmod-yang-metadata-01](#) (work in progress), February 2015.
- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", [RFC 2046](#), November 1996.
- [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.

- [RFC2818] Rescorla, E., "The IETF XML Registry", [RFC 2818](#), May 2000.
- [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.
- [RFC4252] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Authentication Protocol", [RFC 4252](#), January 2006.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", [RFC 5246](#), August 2008.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", [RFC 5277](#), July 2008.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and T. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", [RFC 5280](#), May 2008.
- [RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", [RFC 5789](#), March 2010.
- [RFC5988] Nottingham, M., "Web Linking", [RFC 5988](#), October 2010.
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", [RFC 6020](#), October 2010.
- [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", [RFC 6125](#), March 2011.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", [RFC 6241](#), June 2011.
- [RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", [RFC 6243](#), June 2011.
- [RFC6415] Hammer-Lahav, E. and B. Cook, "Web Host Metadata", [RFC 6415](#), October 2011.

- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", [RFC 6536](#), March 2012.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", [RFC 6570](#), March 2012.
- [RFC6991] Schoenwaelder, J., "Common YANG Data Types", [RFC 6991](#), July 2013.
- [RFC7158] Bray, T., Ed., "The JSON Data Interchange Format", [RFC 7158](#), March 2013.
- [RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", [RFC 7230](#), June 2014.
- [RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", [RFC 7231](#), June 2014.
- [RFC7232] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", [RFC 7232](#), June 2014.
- [RFC7235] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication", [RFC 7235](#), June 2014.
- [RFC7320] Nottingham, M., "URI Design and Ownership", [BCP 190](#), [RFC 7320](#), July 2014.
- [W3C.CR-eventsources-20121211]
Hickson, I., "Server-Sent Events", World Wide Web Consortium CR CR-eventsources-20121211, December 2012, <http://www.w3.org/TR/2012/CR-eventsources-20121211>.
- [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>.
- [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>.

[[draft-ietf-httpauth-basicauth-update-03](#)]

Reschke, J., "The 'Basic' HTTP Authentication Scheme", [draft-ietf-httpauth-basicauth-update-03](#) (work in progress), Dec 2014.

[[draft-ietf-httpauth-digest-09](#)]

Shekh-Yusef, R., Reschke, D., and S. Bremer, "HTTP Digest Access Authentication", [draft-ietf-httpauth-digest-09](#) (work in progress), Dec 2014.

[[draft-ietf-netconf-rfc5539bis-10](#)]

Badra, M., Luchuk, A., and J. Schoenwaelder, "Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication", [draft-ietf-netconf-rfc5539bis-10](#) (work in progress), Dec 2014.

[[draft-thomson-httpbis-cant-01](#)]

Thomson, M., "Client Authentication over New TLS Connection", [draft-thomson-httpbis-cant-01](#) (work in progress), Jul 2014.

14.2. Informative References

[I-D.ietf-netconf-yang-patch]

Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", [draft-ietf-netconf-yang-patch-03](#) (work in progress), January 2015.

[rest-dissertation]

Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000.

Appendix A. Change Log

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

The RESTCONF issue tracker can be found here: <https://github.com/netconf-wg/restconf/issues>

A.1. 04 - 05

- o changed term 'notification event' to 'event notification'
- o removed intro text about framework and meta-model
- o removed early mention of API resources

- o removed term unified datastore and cleaned up text about NETCONF datastores
- o removed text about not immediate persistence of edits
- o removed RESTCONF-specific data-resource-identifier typedef and its usage
- o clarified encoding of key leafs
- o changed several examples from JSON to XML encoding
- o made 'insert' and 'point' query parameters mandatory to implement
- o removed ":insert" capability URI
- o renamed stream/encoding to stream/access
- o renamed stream/encoding/type to stream/access/encoding
- o renamed stream/encoding/events to stream/access/location
- o changed XPath from informative to normative reference
- o changed rest-dissertation from normative to informative reference
- o changed example-jukebox playlist 'id' from a data-resource-identifier to a leafref pointing at the song name

A.2. 03 - 04

- o renamed 'select' to 'fields' (#1)
- o moved collection resource and page capability to [draft-ietf-netconf-restconf-collection-00](#) (#3)
- o added mandatory "defaults" protocol capability URI (#4)
- o added optional "with-defaults" query parameter URI (#4)
- o clarified authentication procedure (#9)
- o moved ietf-yang-library module to [draft-ietf-netconf-yang-library-00](#) (#13)
- o clarified that JSON encoding of module name in a URI MUST follow the netmod-yang-json encoding rules (#14)

- o added restconf-media-type extension (#15)
- o remove "content" query parameter URI and made this parameter mandatory (#16)
- o clarified datastore usage
- o changed lock-denied error example
- o added with-defaults query parameter example
- o added term "RESTCONF Capability"
- o changed NETCONF Capability URI registry usage to new RESTCONF Capability URI Registry usage

A.3. 02 - 03

- o added collection resource
- o added "page" query parameter capability
- o added "limit" and "offset" query parameters, which are available if the "page" capability is supported
- o added "stream list" term
- o fixed bugs in some examples
- o added "encoding" list within the "stream" list to allow different <events> URLs for XML and JSON encoding.
- o made XML MUST implement and JSON MAY implement for servers
- o re-add JSON notification examples (previously removed)
- o updated JSON references

A.4. 01 - 02

- o moved query parameter definitions from the YANG module back to the plain text sections
- o made all query parameters optional to implement
- o defined query parameter capability URI
- o moved 'streams' to new YANG module (ietf-restconf-monitoring)

- o added 'capabilities' container to new YANG module (ietf-restconf-monitoring)
- o moved 'modules' container to new YANG module (ietf-yang-library)
- o added new leaf 'module-set-id' (ietf-yang-library)
- o added new leaf 'conformance' (ietf-yang-library)
- o changed 'schema' leaf to type inet:uri that returns the location of the YANG schema (instead of returning the schema directly)
- o changed 'events' leaf to type inet:uri that returns the location of the event stream resource (instead of returning events directly)
- o changed examples for yang.api resource since the monitoring information is no longer in this resource
- o closed issue #1 'select parameter' since no objections to the proposed syntax
- o closed "encoding of list keys" issue since no objection to new encoding of list keys in a target resource URI.
- o moved open issues list to the issue tracker on github

A.5. 00 - 01

- o fixed content=nonconfig example (non-config was incorrect)
- o closed open issue 'message-id'. There is no need for a message-id field, and [RFC 2392](#) does not apply.
- o closed open issue 'server support verification'. The headers used by RESTCONF are widely supported.
- o removed encoding rules from section on RESTCONF Meta-Data. This is now defined in "I-D.lhotka-netmod-yang-json".
- o added media type application/yang.errors to map to errors YANG grouping. Updated error examples to use new media type.
- o closed open issue 'additional datastores'. Support may be added in the future to identify new datastores.
- o closed open issue 'PATCH media type discovery'. The section on PATCH has an added sentence on the Accept-Patch header.

- o closed open issue 'YANG to resource mapping'. Current mapping of all data nodes to resources will be used in order to allow mandatory DELETE support. The PATCH operation is optional, as well as the YANG Patch media type.
- o closed open issue '_self links for HATEOAS support'. It was decided that they are redundant because they can be derived from the YANG module for the specific data.
- o added explanatory text for the 'select' parameter.
- o added RESTCONF Path Resolution section for discovering the root of the RESTCONF API using the /.well-known/host-meta.
- o added an "error" media type to for structured error messages
- o added Secure Transport section requiring TLS
- o added Security Considerations section
- o removed all references to "REST-like"

A.6. bierman:restconf-04 to ietf:restconf-00

- o updated open issues section

Appendix B. Open Issues

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

The RESTCONF issues are tracked on github.com:

<https://github.com/netconf-wg/restconf/issues>

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 admin
    | | | | +--rw label?          string
    | | | | +--rw catalogue-number? string
    | | | +--rw song* [name]
    | | | | +--rw name      string
    | | | | +--rw location  string
    | | | | +--rw format?   string
    | | | | +--rw length?   uint32
    | | +--ro artist-count? uint32
    | | +--ro album-count?  uint32
    | | +--ro song-count?   uint32
    +--rw playlist* [name]
    | +--rw name      string
    | +--rw description? string
    | +--rw song* [index]
    | | +--rw index    uint32
    | | +--rw id       leafref
  +--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";

  organization "Example, Inc.";
  contact "support at example.com";
  description "Example Jukebox Data Model Module";
  revision "2015-04-04" {
    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";
  }

  container admin {
    description
      "Administrative information for the album.";

    leaf label {
      type string;
      description "The label that released the album.";
    }
    leaf catalogue-number {
```

```
        type string;
        description "The album's catalogue number.";
    }
}

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 leafref {
                path "/jbox:jukebox/jbox:library/jbox:artist/" +
                    "jbox:album/jbox:song/jbox:name";
            }
            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 8](#) 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 ],
    "operations" : {
      "play" : [ null ]
    }
  }
}
```

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/>
  <operations>
    <play xmlns="https://example.com/ns/example-jukebox"/>
  </operations>
</restconf>
```

D.1.2. Retrieve The Server Module Information

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

```
GET /restconf/data/ietf-yang-library:modules 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
Cache-Control: no-cache
Pragma: no-cache
Last-Modified: Sun, 22 Apr 2012 01:00:14 GMT
Content-Type: application/yang.data+json

{
  "ietf-yang-library:modules": {
    "module": [
      {
        "name" : "foo",
        "revision" : "2012-01-02",
        "schema" : "https://example.com/mymodules/foo/2012-01-02",
        "namespace" : "http://example.com/ns/foo",
        "feature" : [ "feature1", "feature2" ],
        "conformance" : true
      },
      {
        "name" : "foo-types",
        "revision" : "2012-01-05",
        "schema" :
          "https://example.com/mymodules/foo-types/2012-01-05",
        "schema" : [null],
        "namespace" : "http://example.com/ns/foo-types",
        "conformance" : false
      },
      {
        "name" : "bar",
```

```
"revision" : "2012-11-05",
"schema" : "https://example.com/mymodules/bar/2012-11-05",
"namespace" : "http://example.com/ns/bar",
"feature" : [ "bar-ext" ],
"conformance" : true,
"submodule" : [
  {
    "name" : "bar-submod1",
    "revision" : "2012-11-05",
    "schema" :
      "https://example.com/mymodules/bar-submod1/2012-11-05"
  },
  {
    "name" : "bar-submod2",
    "revision" : "2012-11-05",
    "schema" :
      "https://example.com/mymodules/bar-submod2/2012-11-05"
  }
]
}
```

D.1.3. Retrieve The Server Capability Information

In this example the client is retrieving the capability information from the server in JSON format, and the server supports all the RESTCONF query parameters, plus one vendor parameter:

```
GET /restconf/data/ietf-restconf-monitoring:restconf-state/
    capabilities HTTP/1.1
Host: example.com
Accept: application/yang.data+xml
```

The server might respond as follows. The extra whitespace in 'capability' elements for display purposes only.

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:02:00 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.data+xml
```



```
<capabilities xmlns="">
  <capability>
    urn:ietf:params:restconf:capability:content:1.0
  </capability>
  <capability>
    urn:ietf:params:restconf:capability:depth:1.0
  </capability>
  <capability>
    urn:ietf:params:restconf:capability:fields:1.0
  </capability>
  <capability>
    urn:ietf:params:restconf:capability:filter:1.0
  </capability>
  <capability>
    urn:ietf:params:restconf:capability:point:1.0
  </capability>
  <capability>
    urn:ietf:params:restconf:capability:start-time:1.0
  </capability>
  <capability>
    urn:ietf:params:restconf:capability:stop-time:1.0
  </capability>
  <capability>
    http://example.com/capabilities/myparam
  </capability>
</capabilities>
```

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: https://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: https://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
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.2.3. Edit a Datastore Resource

In this example, the client modifies two different data nodes by sending a PATCH to the datastore resource:

```
PATCH /restconf/data HTTP/1.1
Host: example.com
Content-Type: application/yang.datastore+xml

<data xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <jukebox xmlns="http://example.com/ns/example-jukebox">
    <library>
      <artist>
        <name>Foo Fighters</name>
        <album>
          <name>Wasting Light</name>
          <year>2011</year>
        </album>
      </artist>
      <artist>
        <name>Nick Cave</name>
        <album>
          <name>Tender Prey</name>
          <year>1988</year>
        </album>
      </artist>
    </library>
  </jukebox>
</data>
```

D.3. Query Parameter Examples

D.3.1. "content" Parameter

The "content" parameter is used to select the type of data child 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 child 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 child 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 child 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=nonconfig

To retrieve only the non-configuration child resources, the "content" parameter is set to "nonconfig". 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=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 17:11:30 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 child resources that are returned by the server for a GET method request.

The depth parameter starts counting levels at the level of the target resource that is specified, so that a depth level of "1" includes just the target resource level itself. A depth level of "2" includes the target resource level and its child nodes.

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 child 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" : "https://example.com/restconf/data/
            example-jukebox:jukebox/library/artist=
            Foo%20Fighters/album/Wasting%20Light/"
        }
      ]
    }
  ]
}
```



```
        song/Rope"
      },
      {
        "index" : 2,
        "id" : "https://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 child 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. "fields" Parameter

In this example the client is retrieving the API resource, but retrieving only the "name" and "revision" nodes from each module, in JSON format:

```
GET /restconf/data?fields=modules/module(name;revision) 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
```

```
{
  "ietf-yang-library:modules": {
    "module": [
      {
        "name" : "example-jukebox",
        "revision" : "2015-06-04"
      },
      {
        "name" : "ietf-inet-types",
        "revision" : "2013-07-15"
      },
      {
        "name" : "ietf-restconf-monitoring",
        "revision" : "2015-06-04"
      },
      {
        "name" : "ietf-yang-library",
        "revision" : "2015-01-30"
      },
      {
        "name" : "ietf-yang-types",
        "revision" : "2013-07-15"
      }
    ]
  }
}
```

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: https://example.com/restconf/data/
  example-jukebox:jukebox/playlist=Foo-One/song=1
ETag: eeeada438af
```

D.3.5. "point" Parameter

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

1. 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. "filter" Parameter

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

```
// filter = /event/event-class='fault'
GET /mystreams/NETCONF?filter=%2Fevent%2Fevent-class%3D'fault'

// filter = /event/severity<=4
GET /mystreams/NETCONF?filter=%2Fevent%2Fseverity%3C%3D4

// filter = /linkUp|/linkDown
GET /mystreams/SNMP?filter=%2FlinkUp%7C%2FlinkDown

// filter = /*/reporting-entity/card!='Ethernet0'
GET /mystreams/NETCONF?
  filter=%2F*%2Freporting-entity%2Fcard%21%3D'Ethernet0'

// filter = /*/email-addr[contains(.,'company.com')]
GET /mystreams/critical-syslog?
  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 /mystreams/NETCONF?
  filter=(%2Fexample-mod%3Aevent1%2Fname%3D'joe'%20and
          %20%2Fexample-mod%3Aevent1%2Fstatus%3D'online')
```

D.3.7. "start-time" Parameter

```
// start-time = 2014-10-25T10:02:00Z
GET /mystreams/NETCONF?start-time=2014-10-25T10%3A02%3A00Z
```

D.3.8. "stop-time" Parameter

```
// stop-time = 2014-10-25T12:31:00Z
GET /mystreams/NETCONF?stop-time=2014-10-25T12%3A31%3A00Z
```

D.3.9. "with-defaults" Parameter

Assume the same data model as defined in [Appendix A.1 of \[RFC6243\]](#). Assume the same data set as defined in [Appendix A.2 of \[RFC6243\]](#). If the server defaults-uri basic-mode is "trim", the the following request for interface "eth1" might be as follows:

Without query parameter:

```
GET /restconf/data/interfaces/interface=eth1 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:interface": [
    {
      "name" : "eth1",
      "status" : "up"
    }
  ]
}
```

Note that the "mtu" leaf is missing because it is set to the default "1500", and the server defaults handling basic-mode is "trim".

With query parameter:

```
GET /restconf/data/interfaces/interface=eth1
    ?with-defaults=report-all 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:interface": [
    {
      "name" : "eth1",
      "mtu" : 1500,
      "status" : "up"
    }
  ]
}
```

Note that the server returns the "mtu" leaf because the "report-all" mode was requested with the "with-defaults" query parameter.

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