

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 1, 2017

A. Bierman
YumaWorks
M. Bjorklund
Tail-f Systems
K. Watsen
Juniper Networks
September 28, 2016

RESTCONF Protocol
draft-ietf-netconf-restconf-17

Abstract

This document describes an HTTP-based protocol that provides a programmatic interface for accessing data defined in YANG, using the datastore concepts 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 April 1, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in [Section 4.e](#) of

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

Table of Contents

- [1. Introduction](#) [5](#)
- [1.1. Terminology](#) [6](#)
- [1.1.1. NETCONF](#) [6](#)
- [1.1.2. HTTP](#) [6](#)
- [1.1.3. YANG](#) [7](#)
- [1.1.4. NETCONF Notifications](#) [7](#)
- [1.1.5. Terms](#) [8](#)
- [1.1.6. URI Template and Examples](#) [10](#)
- [1.1.7. Tree Diagrams](#) [10](#)
- [1.2. Subset of NETCONF Functionality](#) [11](#)
- [1.3. Data Model Driven API](#) [11](#)
- [1.4. Coexistence with NETCONF](#) [12](#)
- [1.5. RESTCONF Extensibility](#) [13](#)
- [2. Transport Protocol Requirements](#) [15](#)
- [2.1. Integrity and Confidentiality](#) [15](#)
- [2.2. HTTPS with X.509v3 Certificates](#) [15](#)
- [2.3. Certificate Validation](#) [15](#)
- [2.4. Authenticated Server Identity](#) [15](#)
- [2.5. Authenticated Client Identity](#) [16](#)
- [3. Resources](#) [16](#)
- [3.1. Root Resource Discovery](#) [17](#)
- [3.2. RESTCONF Media Types](#) [19](#)
- [3.3. API Resource](#) [19](#)
- [3.3.1. {+restconf}/data](#) [20](#)
- [3.3.2. {+restconf}/operations](#) [20](#)
- [3.3.3. {+restconf}/yang-library-version](#) [21](#)
- [3.4. Datastore Resource](#) [21](#)
- [3.4.1. Edit Collision Prevention](#) [22](#)
- [3.5. Data Resource](#) [23](#)
- [3.5.1. Timestamp](#) [24](#)
- [3.5.2. Entity-Tag](#) [24](#)
- 3.5.3. Encoding Data Resource Identifiers in the Request URI [24](#)
- [3.5.4. Default Handling](#) [28](#)
- [3.6. Operation Resource](#) [29](#)
- [3.6.1. Encoding Operation Resource Input Parameters](#) [30](#)
- [3.6.2. Encoding Operation Resource Output Parameters](#) [34](#)
- [3.6.3. Encoding Operation Resource Errors](#) [36](#)
- [3.7. Schema Resource](#) [37](#)
- [3.8. Event Stream Resource](#) [38](#)
- [3.9. Errors YANG Data Template](#) [39](#)
- [4. RESTCONF Methods](#) [39](#)
- [4.1. OPTIONS](#) [40](#)
- [4.2. HEAD](#) [40](#)

4.3.	GET	40
4.4.	POST	42
4.4.1.	Create Resource Mode	42
4.4.2.	Invoke Operation Mode	44
4.5.	PUT	44
4.6.	PATCH	46
4.6.1.	Plain Patch	47
4.7.	DELETE	48
4.8.	Query Parameters	49
4.8.1.	The "content" Query Parameter	50
4.8.2.	The "depth" Query Parameter	51
4.8.3.	The "fields" Query Parameter	51
4.8.4.	The "filter" Query Parameter	52
4.8.5.	The "insert" Query Parameter	53
4.8.6.	The "point" Query Parameter	54
4.8.7.	The "start-time" Query Parameter	54
4.8.8.	The "stop-time" Query Parameter	55
4.8.9.	The "with-defaults" Query Parameter	55
5.	Messages	57
5.1.	Request URI Structure	57
5.2.	Message Encoding	58
5.3.	RESTCONF Metadata	59
5.3.1.	XML Metadata Encoding Example	60
5.3.2.	JSON Metadata Encoding Example	60
5.4.	Return Status	61
5.5.	Message Caching	61
6.	Notifications	62
6.1.	Server Support	62
6.2.	Event Streams	62
6.3.	Subscribing to Receive Notifications	64
6.3.1.	NETCONF Event Stream	65
6.4.	Receiving Event Notifications	65
7.	Error Reporting	67
7.1.	Error Response Message	68
8.	RESTCONF Module	70
9.	RESTCONF Monitoring	77
9.1.	restconf-state/capabilities	77
9.1.1.	Query Parameter URIs	78
9.1.2.	The "defaults" Protocol Capability URI	78
9.2.	restconf-state/streams	79
9.3.	RESTCONF Monitoring Module	79
10.	YANG Module Library	83
10.1.	modules-state/module	83
11.	IANA Considerations	83
11.1.	The "restconf" Relation Type	84
11.2.	YANG Module Registry	84
11.3.	Media Types	85
11.3.1.	Media Type application/yang-data+xml	85

11.3.2 . Media Type application/yang-data+json	86
11.4 . RESTCONF Capability URNs	88
12 . Security Considerations	89
13 . Acknowledgements	90
14 . References	90
14.1 . Normative References	90
14.2 . Informative References	93
Appendix A . Change Log	93
A.1 . v16 to v17	93
A.2 . v15 to v16	94
A.3 . v14 to v15	94
A.4 . v13 - v14	95
A.5 . v12 - v13	96
A.6 . v11 - v12	97
A.7 . v10 - v11	97
A.8 . v09 - v10	98
A.9 . v08 - v09	100
A.10 . v07 - v08	100
A.11 . v06 - v07	100
A.12 . v05 - v06	101
A.13 . v04 - v05	101
A.14 . v03 - v04	102
A.15 . v02 - v03	102
A.16 . v01 - v02	103
A.17 . v00 - v01	104
A.18 . bierman:restconf-04 to ietf:restconf-00	105
Appendix B . Open Issues	105
Appendix C . Example YANG Module	105
C.1 . example-jukebox YANG Module	106
Appendix D . RESTCONF Message Examples	111
D.1 . Resource Retrieval Examples	111
D.1.1 . Retrieve the Top-level API Resource	111
D.1.2 . Retrieve The Server Module Information	112
D.1.3 . Retrieve The Server Capability Information	114
D.2 . Edit Resource Examples	115
D.2.1 . Create New Data Resources	115
D.2.2 . Detect Resource Entity-Tag Change	117
D.2.3 . Edit a Datastore Resource	117
D.2.4 . Edit a Data Resource	118
D.3 . Query Parameter Examples	118
D.3.1 . "content" Parameter	119
D.3.2 . "depth" Parameter	122
D.3.3 . "fields" Parameter	125
D.3.4 . "insert" Parameter	126
D.3.5 . "point" Parameter	127
D.3.6 . "filter" Parameter	128
D.3.7 . "start-time" Parameter	129
D.3.8 . "stop-time" Parameter	129

[D.3.9.](#) "with-defaults" Parameter [130](#)
 Authors' Addresses [131](#)

1. Introduction

There is a need for standard mechanisms to allow Web applications to access the configuration data, state data, data-model-specific RPC operations, and event notifications within a networking device, in a modular and extensible manner.

This document defines an HTTP [[RFC7230](#)] based protocol called RESTCONF, for configuring data defined in YANG version 1 [[RFC6020](#)] or YANG version 1.1 [[RFC7950](#)], using the datastore concepts defined in NETCONF [[RFC6241](#)].

NETCONF defines configuration datastores and a set of Create, Retrieve, Update, Delete (CRUD) operations that can be used to access these datastores. NETCONF also defines a protocol for invoking these operations. The YANG language defines the syntax and semantics of datastore content, configuration, state data, RPC operations, and event notifications.

RESTCONF uses HTTP methods to provide CRUD operations on a conceptual datastore containing YANG-defined data, which is compatible with a server which implements NETCONF datastores.

If a RESTCONF server is co-located with a NETCONF server, then there are protocol interactions with the NETCONF protocol, which are described in [Section 1.4](#). The RESTCONF server MAY provide access to specific datastores using operation resources, as described in [Section 3.6](#). The RESTCONF protocol does not specify any mandatory operation resources. The semantics of each operation resource determine if and how datastores are accessed.

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 [[RFC7159](#)].

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

1.1. Terminology

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

1.1.1. NETCONF

The following terms are defined in [[RFC6241](#)]:

- o candidate configuration datastore
- o configuration data
- o datastore
- o configuration datastore
- o running configuration datastore
- o startup configuration datastore
- o state data
- o user

1.1.2. HTTP

The following terms are defined in [[RFC3986](#)]:

- o fragment
- o path
- o query

The following terms are defined in [[RFC7230](#)]:

- o header field
- 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

1.1.3. YANG

The following terms are defined in [[RFC7950](#)]:

- o action
- o container
- o data node
- o key leaf
- o leaf
- o leaf-list
- o list
- o mandatory node
- o ordered-by user
- o presence container
- o RPC operation
- o top-level data node

1.1.4. NETCONF Notifications

The following terms are defined in [[RFC5277](#)]:

- o notification replay

[1.1.5.](#) Terms

The following terms are used within this document:

- o API resource: the resource that models the RESTCONF root resource and the sub-resources to access YANG-defined content. It is defined with the YANG data template named "yang-api" in the "ietf-restconf" module.
- o client: a RESTCONF client
- o data resource: a resource that models a YANG data node. It is defined with YANG data definition statements.
- o datastore resource: the resource that models a programmatic interface using NETCONF datastore concepts. By default, RESTCONF methods access a unified view of the underlying datastore implementation on the server. It is defined as a sub-resource within the API resource.
- o edit operation: a RESTCONF operation on a data resource using either a POST, PUT, PATCH, or DELETE method. This is not the same as the NETCONF edit operation (i.e., one of the values for the "nc:operation" attribute: "create", "replace", "merge", "delete", or "remove").
- 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 SSE [[W3C.REC-eventsource-20150203](#)] specification. Event stream contents are described in [Section 3.8](#).
- 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 NETCONF client: a client which implements the NETCONF protocol. Called "client" in [[RFC6241](#)].
- o NETCONF server: a server which implements the NETCONF protocol. Called "server" in [[RFC6241](#)].
- o operation: the conceptual RESTCONF operation for a message, derived from the HTTP method, request URI, header fields, and message-body.

- o operation resource: a resource that models a data-model-specific operation, that is defined with a YANG "rpc" or "action" statement. It is invoked with the POST method.
- o patch: a PATCH method 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 media type for use with the PATCH method, defined in [Section 4.6.1](#), that can be used for simple merge operations. It is specified by a request Content-Type of "application/yang-data+xml" or "application/yang-data+json".
- o query parameter: a parameter (and its value if any), encoded within the query component of the request URI.
- o resource type: one of the RESTCONF resource classes defined in this document. One of "api", "datastore", "data", "operation", "schema", or "event stream".
- 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 defined in [Section 9.3](#).
- o RESTCONF client: a client which implements the RESTCONF protocol.
- o RESTCONF server: a server which implements the RESTCONF protocol.
- o retrieval request: a request using the GET or HEAD methods.
- o schema resource: a resource that used by the client to retrieve a YANG schema with the GET method. It has a representation with the media type "application/yang".
- o server: a RESTCONF server
- 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.
- o stream resource: An event stream resource.

- o target resource: the resource that is associated with a particular message, identified by the "path" component of the request URI.
- o yang-data extension: A YANG external statement that conforms to the "yang-data" extension statement found in [Section 8](#). The yang-data extension is used to define YANG data structures that are meant to be used as YANG data templates. These data structures are not intended to be implemented as part of a configuration datastore or as operational state within the server, so normal YANG data definition statements cannot be used.
- o YANG data template: a schema for modeling protocol message components as conceptual data structure using YANG. This allows the messages to be defined in an encoding-independent manner. Each YANG data template is defined with the "yang-data" extension, found in [Section 8](#). Representations of instances conforming to a particular YANG data template can be defined for YANG. The XML representation is defined in YANG version 1.1 [[RFC7950](#)], and supported with the "application/yang-data+xml" media type. The JSON representation is defined in JSON Encoding of Data Modeled with YANG [[RFC7951](#)], and supported with the "application/yang-data+json" media type.

[1.1.6](#). URI Template and Examples

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

For simplicity, all of the examples in this document use "/restconf" as the discovered RESTCONF API root path. Many of the examples throughout the document are based on the "example-jukebox" YANG module, defined in [Appendix C.1](#).

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

[1.1.7](#). Tree 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), "ro" state data (read-only), and "x" operation resource (executable)
- 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.

1.2. Subset of NETCONF Functionality

RESTCONF does not need to mirror the full functionality of the NETCONF protocol, but it does need to be compatible with NETCONF. RESTCONF achieves this by implementing a subset of the interaction capabilities provided by the NETCONF protocol, for instance, by eliminating datastores and explicit locking.

RESTCONF uses HTTP methods to implement the equivalent of NETCONF operations, enabling basic CRUD operations on a hierarchy of conceptual resources.

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 by a RESTCONF client.

RESTCONF is not intended to replace NETCONF, but rather provide an HTTP interface that follows Representational State Transfer (REST) principles [[rest-dissertation](#)], and is compatible with the NETCONF datastore model.

1.3. Data Model Driven API

RESTCONF combines the simplicity of the HTTP protocol with the predictability and automation potential of a schema-driven API. Knowing the YANG modules used by the server, a client can derive all management resource URLs and the proper structure of all RESTCONF requests and responses. This strategy obviates the need for responses provided by the server to contain Hypermedia as the Engine of Application State (HATEOAS) links, originally described in Roy Fielding's doctoral dissertation [[rest-dissertation](#)], because the client can determine the links it needs from the YANG modules.

RESTCONF utilizes the YANG Library [[RFC7895](#)] to allow a client to discover the YANG module conformance information for the server, in case the client wants to use it.

The server can optionally support retrieval of the YANG modules it uses, as identified in its YANG library. See [Section 3.7](#) for details.

The URIs for data-model-specific RPC operations and datastore content are predictable, based on the YANG module definitions.

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 [[RFC7895](#)]. The server MUST implement the "ietf-yang-library" module, which MUST identify all the YANG modules used by the server, in the "modules-state/module" list. The conceptual datastore contents, data-model-specific RPC operations and event notifications are identified by this set of YANG modules.

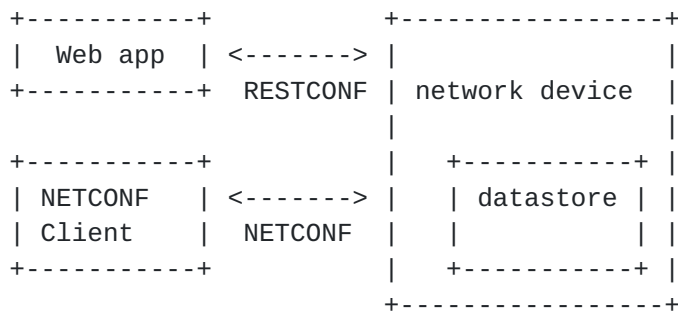
The classification of data as configuration or non-configuration is derived from the YANG "config" statement. Data ordering behavior is derived from the YANG "ordered-by" statement. Non-configuration data is also called "state data".

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 data resource within the datastore resource is activated upon successful completion of the edit.

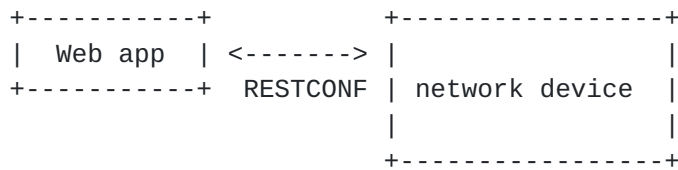
1.4. Coexistence with NETCONF

RESTCONF can be implemented on a device that supports the NETCONF protocol.

The following figure shows the system components if a RESTCONF server is co-located with a NETCONF server:



The following figure shows the system components if a RESTCONF server is implemented in a device that does not have a NETCONF server:



Note that there are no interactions at all between the NETCONF protocol and RESTCONF protocol. It is possible that locks are in use on a RESTCONF server, even though RESTCONF cannot manipulate locks. In such a case, the RESTCONF protocol will not be granted write access to data resources within a datastore.

If the NETCONF server supports `:writable-running`, all edits to configuration nodes in `{+restconf}/data` are performed in the running configuration datastore. The URI template `"{+restconf}"` is defined in [Section 1.1.6](#).

Otherwise, if the device supports `:candidate`, all edits to configuration nodes in `{+restconf}/data` are performed in the candidate configuration datastore. The candidate MUST be automatically committed to running immediately after each successful edit. Any edits from other sources that are in the candidate datastore will also be committed. If a confirmed commit procedure is in progress by any NETCONF client, then this commit will act as the confirming commit. If the NETCONF server is expecting a `"persist-id"` parameter to complete the confirmed commit procedure then the RESTCONF edit operation MUST fail with a `"409 Conflict"` status-line. The error-tag `"in-use"` is returned in this case. The error-tag value `"resource-denied"` is used in this case.

If the NETCONF server supports `:startup`, the RESTCONF server MUST automatically update the non-volatile startup configuration datastore, after the running datastore has been altered as a consequence of a RESTCONF edit operation.

If a datastore that would be modified by a RESTCONF operation has an active lock from a NETCONF client, the RESTCONF edit operation MUST fail with a `"409 Conflict"` status-line. The error-tag `"in-use"` is returned in this case.

1.5. RESTCONF Extensibility

There are two extensibility mechanisms built into RESTCONF:

- o protocol version

- o optional capabilities

This document defines version 1 of the RESTCONF protocol. If a future version of this protocol is defined, then that document will specify how the new version of RESTCONF is identified. It is expected that a different RESTCONF root resource will be used which will be located using a different link relation (See [Section 3.1](#)).

The server will advertise all protocol versions that it supports in its host-meta data.

In this example, the server supports both RESTCONF version 1 and a fictitious version 2.

The client might send:

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

The server might respond:

```
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'/>
  <Link rel='restconf2' href='/restconf2'/>
</XRD>
```

RESTCONF also supports a server-defined list of optional capabilities, which are listed by a server using the "ietf-restconf-monitoring" module defined in [Section 9.3](#). This document defines several query parameters in [Section 4.8](#). Each optional parameter has a corresponding capability URI defined in [Section 9.1.1](#) that is advertised by the server if supported.

The "capabilities" list can identify any sort of server extension. Currently this extension mechanism is used to identify optional query parameters that are supported, but it is not limited to that purpose. For example, the "defaults" URI defined in [Section 9.1.2](#) specifies a mandatory URI identifying server defaults handling behavior.

A new sub-resource type could be identified with a capability if it is optional to implement. Mandatory protocol features and new resource types require a new revision of the RESTCONF protocol.

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. A RESTCONF server MUST support the TLS protocol [[RFC5246](#)]. The RESTCONF protocol MUST NOT be used over HTTP without using the TLS protocol.

RESTCONF does not require a specific version of HTTP. However, it is RECOMMENDED that at least HTTP/1.1 [[RFC7230](#)] be supported by all implementations.

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.

RESTCONF servers MUST present an X.509v3 based certificate when establishing a TLS connection with a RESTCONF client. The use of X.509v3 based certificates is consistent with NETCONF over TLS [[RFC7589](#)].

2.3. Certificate Validation

The RESTCONF client MUST either use X.509 certificate path validation [[RFC5280](#)] to verify the integrity of the RESTCONF server's TLS certificate, or match the server's TLS certificate with a certificate obtained by a trusted mechanism (e.g. a pinned certificate). If X.509 certificate path validation fails, and the presented X.509 certificate does not match a certificate obtained by a trusted mechanism, the connection MUST be terminated, as described in [Section 7.2.1 of \[\[RFC5246\]\(#\)\]](#).

2.4. Authenticated Server Identity

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. If the RESTCONF client is not authenticated, the server SHOULD send an HTTP response with "401 Unauthorized" status-line, as defined in [Section 3.1 of \[RFC7235\]](#). The error-tag value "access-denied" is used in this case.

To authenticate a client, a RESTCONF server MUST use TLS based client certificates ([Section 7.4.6 of \[RFC5246\]](#)), or MUST use any HTTP authentication scheme defined in the HTTP Authentication Scheme Registry ([Section 5.1 in \[RFC7235\]](#)). A server MAY also support the combination of both client certificates and an HTTP client authentication scheme, with the determination of how to process this combination left as an implementation decision.

The RESTCONF client identity derived from the authentication mechanism used is hereafter known as the "RESTCONF username" and subject to the NETCONF Access Control Module (NACM) [[RFC6536](#)]. When a client certificate is presented, the RESTCONF username MUST be derived using the algorithm defined in [Section 7 of \[RFC7589\]](#). For all other cases, when HTTP authentication is used, the RESTCONF username MUST be provided by the HTTP authentication scheme used.

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 as a collection of 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 a representation associated with a media type identifier, as represented by the "Content-Type" header field in the HTTP response message. A resource has one or more representations, each associated with a different media type. When a representation of a resource is sent in an HTTP message, the associated media type is given in the "Content-Type" header. 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.

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 RPC operations, top-level data nodes, and event notification messages supported by the server.

The RESTCONF protocol does not include a data resource discovery mechanism. Instead, the definitions within the YANG modules advertised by the server are used to construct an RPC 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. There MUST be exactly one "restconf" link relation returned by the device.

The client discovers this by getting the `"/.well-known/host-meta"` resource ([\[RFC6415\]](#)) and using the `<Link>` element containing the "restconf" attribute :

Example returning `/restconf`:

The client might send:

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

The server might respond:

```
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>
```

After discovering the RESTCONF API root, the client MUST use this value as the initial part of the path in the request URI, in any subsequent request for a RESTCONF resource.

In this example, the client would use the path `"/restconf"` as the RESTCONF root resource.

Example returning `/top/restconf`:

The client might send:


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

The server might respond:

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

In this example, the client would use the path `"/top/restconf"` as the RESTCONF root resource.

The client can now determine the operation resources supported by the the server. In this example a custom `"play"` operation is supported:

The client might send:

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

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2016 17:01:00 GMT
Server: example-server
Cache-Control: no-cache
Last-Modified: Sun, 22 Apr 2016 01:00:14 GMT
Content-Type: application/yang-data+json

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

If the Extensible Resource Descriptor (XRD) contains more than one link relation, then only the relation named `"restconf"` is relevant to this specification.

Note that any given endpoint (host:port) can only support one RESTCONF server, due to the root resource discovery mechanism. This limits the number of RESTCONF servers that can run concurrently on a host, since each server must use a different port.

[3.2.](#) RESTCONF Media Types

The RESTCONF protocol defines two application specific media types to identify representations of data which conforms to the schema for a particular YANG construct.

This document defines media types for XML and JSON serialization of YANG data. Other documents MAY define other media types for different serializations of YANG data. The "application/yang-data+xml" media-type is defined in [Section 11.3.1](#). The "application/yang-data+json" media-type is defined in [Section 11.3.2](#).

[3.3.](#) API Resource

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

YANG Tree Diagram for an API Resource:

```
+---- {+restconf}
  +---- data
  | ...
  +---- operations?
  | ...
  +--ro yang-library-version    string
```

The "yang-api" YANG data template is defined using the "yang-data" extension in the "ietf-restconf" module, found in [Section 8](#). It specifies the structure and syntax of the conceptual child resources within the API resource.

The API resource can be retrieved with the GET method.

The {+restconf} root resource name used in responses representing the root of the "ietf-restconf" module MUST identify the "ietf-restconf" YANG module. For example, a request to GET the root resource "/restconf" in JSON format will return a representation of the API resource named "ietf-restconf:restconf".

This resource has the following child resources:

Child Resource	Description
data	Contains all data resources
operations	Data-model-specific operations
yang-library-version	ietf-yang-library module date

RESTCONF API Resource

[3.3.1.](#) `{+restconf}/data`

This mandatory resource represents the combined configuration and state 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 2016 17:01:30 GMT
Server: example-server
Cache-Control: 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 RPC operations supported by the server. The server MAY omit this resource if no data-model-specific RPC operations are advertised.

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

The access point for each RPC operation is represented as an empty leaf. If an operation resource is retrieved, the empty leaf representation is returned by the server.

Operation resources are defined in [Section 3.6](#).

[3.3.3](#). `{+restconf}/yang-library-version`

This mandatory leaf identifies the revision date of the "ietf-yang-library" YANG module that is implemented by this server. Note that the revision date for the module version found in [[RFC7895](#)] is used.

Example:

```
GET /restconf/yang-library-version HTTP/1.1
Host: example.com
Accept: application/yang-data+xml
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2016 17:01:30 GMT
Server: example-server
Cache-Control: no-cache
Content-Type: application/yang-data+xml

<yang-library-version
  xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">\
  2016-06-21\
</yang-library-version>
```

[3.4](#). `Datastore Resource`

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

This resource type is an abstraction of the system's underlying datastore implementation. The client uses it to edit and retrieve data resources, as the conceptual root of all configuration and state 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 be written directly with the POST and PATCH methods. Each RESTCONF edit of a datastore resource is saved to non-volatile storage by the server, if the server supports non-volatile storage of configuration data, as described in [Section 1.4](#).

If the datastore resource represented by the "{+restconf}/data" subtree is retrieved, then the datastore and its contents are returned by the server. The datastore is represented by a node named "data" in the "ietf-restconf" module namespace.

[3.4.1](#). Edit Collision Prevention

Two edit collision detection and prevention mechanisms are provided in RESTCONF for the datastore resource: a timestamp and an entity-tag. Any change to configuration data resources updates the timestamp and entity tag of the datastore resource. In addition, the RESTCONF server MUST return an error if the datastore is locked by an external source (e.g., NETCONF server).

[3.4.1.1](#). Timestamp

The last change time is maintained and the "Last-Modified" ([\[RFC7232\]](#), [Section 2.2](#)) header field is returned in the response for a retrieval request. The "If-Unmodified-Since" header field ([\[RFC7232\]](#), [Section 3.4](#)) 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 SHOULD maintain a last-modified timestamp for the datastore resource, defined in [Section 3.4](#). This timestamp is only affected by configuration child data resources, and MUST NOT be updated for changes to non-configuration child data resources. Last-modified timestamps for data resources are discussed in [Section 3.5](#).

If the RESTCONF server is colocated with a NETCONF server, then the last-modified timestamp MUST be for the "running" datastore. Note that it is possible other protocols can cause the last-modified timestamp to be updated. Such mechanisms are out of scope for this document.

[3.4.1.2](#). Entity-Tag

The server MUST maintain a unique opaque entity-tag for the datastore resource and MUST return it in the "ETag" ([\[RFC7232\]](#), [Section 2.3](#)) header in the response for a retrieval request. The client MAY use an "If-Match" header 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. This entity-tag is only affected by configuration data resources, and MUST NOT be updated for changes to non-configuration data. Entity-tags for data resources are discussed in [Section 3.5](#). Note that each representation (e.g. XML vs. JSON) requires a different entity-tag.

If the RESTCONF server is colocated with a NETCONF server, then this entity-tag MUST be for the "running" datastore. Note that it is possible other protocols can cause the entity-tag to be updated. Such mechanisms are out of scope for this document.

[3.4.1.3](#). Update Procedure

Changes to configuration data resources affect the timestamp and entity-tag for that resource, any ancestor data resources, and the datastore resource.

For example, an edit to disable an interface might be done by setting the leaf `/interfaces/interface/enabled` to `false`. The `enabled` data node and its ancestors (one `interface` list instance, and the `interfaces` container) are considered to be changed. The datastore is considered to be changed when any top-level configuration data node is changed (e.g., `interfaces`).

[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 method. Containers, leafs, leaf-list entries, list entries, anydata and anyxml nodes are data resources.

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

A data resource can be retrieved with the GET method. Data resources are accessed via the `{+restconf}/data` URI. This sub-tree is used to retrieve and edit data resources.

3.5.1. Timestamp

For configuration data resources, the server MAY maintain a last-modified timestamp for the resource, and return the "Last-Modified" header field when it is retrieved with the GET or HEAD methods.

The "Last-Modified" header field can be used by a RESTCONF client in subsequent requests, within the "If-Modified-Since" and "If-Unmodified-Since" header fields.

If maintained, the resource timestamp MUST be set to the current time whenever the resource or any configuration resource within the resource is altered. If not maintained, then the resource timestamp for the datastore MUST be used instead. If the RESTCONF server is colocated with a NETCONF server, then the last-modified timestamp for a configuration data resource MUST represent the instance within the "running" datastore.

This timestamp is only affected by configuration data resources, and MUST NOT be updated for changes to non-configuration data.

3.5.2. Entity-Tag

For configuration data resources, the server SHOULD maintain a resource entity-tag for each resource, and return the "ETag" header field 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. If not maintained, then the resource entity-tag for the datastore MUST be used instead.

The "ETag" header field can be used by a RESTCONF client in subsequent requests, within the "If-Match" and "If-None-Match" header fields.

This entity-tag is only affected by configuration data resources, and MUST NOT be updated for changes to non-configuration data. If the RESTCONF server is colocated with a NETCONF server, then the entity-tag for a configuration data resource MUST represent the instance within the "running" datastore.

3.5.3. Encoding Data Resource Identifiers in the Request URI

In YANG, data nodes can be identified with an absolute XPath expression, defined in [[XPath](#)], starting from the document root to the target resource. In RESTCONF, URI-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 encoded from left to right, starting with the top-level data node, according to the "api-path" rule in [Section 3.5.3.1](#). 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 node in the path is defined in another module than its parent node, or its parent is the datastore, then the module name followed by a colon character (":") MUST be prepended to the node name in the resource identifier. See [Section 3.5.3.1](#) for details.

If a data node in the path expression is a YANG leaf-list node, then the leaf-list value MUST be encoded according to the following rules:

- o The identifier for the leaf-list MUST be encoded using one path segment [[RFC3986](#)].
- o The path segment is constructed by having the leaf-list name, followed by an "=" character, followed by the leaf-list value. (e.g., /restconf/data/top-leaflist=fred).
- o The leaf-list value is specified as a string, using the canonical representation for the YANG data type. Any reserved characters MUST be percent-encoded, according to [[RFC3986](#)], [section 2.1](#) and 2.5.
- o YANG 1.1 allows duplicate leaf-list values for non-configuration data. In this case there is no mechanism to specify the exact matching leaf-list instance.
- o The comma (',') character is percent-encoded, even though multiple key values are not possible for a leaf-list. This is more consistent and avoids special processing rules.

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 "=" character, followed by the single key leaf value.

- o If there are multiple key leaf values, the path segment is constructed by having the list name, followed by the value of each leaf identified in the "key" statement, encoded in the order specified in the YANG "key" statement. Each key leaf value except the last one is followed by a comma character.
- o The key value is specified as a string, using the canonical representation for the YANG data type. Any reserved characters MUST be percent-encoded, according to [\[RFC3986\], section 2.1](#) and 2.5. The comma (',') character MUST be percent-encoded if it is present in the key value.
- o All the components in the "key" statement MUST be encoded. Partial instance identifiers are not supported.
- o Missing key values are not allowed, so two consecutive commas are interpreted as a comma, followed by a zero-length string, followed by a comma. For example, "list1=foo,,baz" would be interpreted as a list named "list1" with 3 key values, and the second key value is a zero-length string.
- o Note that non-configuration lists are not required to define keys. In this case, a single list instance cannot be accessed.
- o The "list-instance" ABNF rule defined in [Section 3.5.3.1](#) represents the syntax of a list instance identifier.

Examples:

```
container top {
  list list1 {
    key "key1 key2 key3";
    ...
    list list2 {
      key "key4 key5";
      ...
      leaf X { type string; }
    }
  }
  leaf-list Y {
    type uint32;
  }
}
```

For the above YANG definition, the container "top" is defined in the "example-top" YANG module, and a target resource URI for leaf "X" would be encoded as follows:


```
/restconf/data/example-top:top/list1=key1,key2,key3/\
list2=key4,key5/X
```

For the above YANG definition, a target resource URI for leaf-list "Y" would be encoded as follows:

```
/restconf/data/example-top:top/Y=instance-value
```

The following example shows how reserved characters are percent-encoded within a key value. The value of "key1" contains a comma, single-quote, double-quote, colon, double-quote, space, and forward slash. (, ' " : " /). Note that double-quote is not a reserved character and does not need to be percent-encoded. The value of "key2" is the empty string, and the value of "key3" is the string "foo".

Example URL:

```
/restconf/data/example-top:top/list1=%2C%27"%3A"%20%2F,,foo
```

3.5.3.1. ABNF For Data Resource Identifiers

The "api-path" Augmented Backus-Naur Form (ABNF) syntax is used to construct RESTCONF path identifiers. Note that this syntax is used for all resources, and the API path starts with the RESTCONF root resource. Data resources are required to be identified under the subtree "+{restconf}/data".

An identifier is not allowed to start with the case-insensitive string "XML", according to YANG identifier rules. The syntax for "api-identifier" and "key-value" MUST conform to the JSON identifier encoding rules in [Section 4 of \[RFC7951\]](#): The RESTCONF root resource path is required. Additional sub-resource identifiers are optional. The characters in a key value string are constrained, and some characters need to be percent-encoded, as described in [Section 3.5.3](#).


```

api-path = root ["/" (api-identifier | list-instance)]*
root = string ;; replacement string for {+restconf}
api-identifier = [module-name ":" ] identifier
module-name = identifier
list-instance = api-identifier "=" key-value ["," key-value]*
key-value = string ;; constrained chars are percent-encoded
string = <an unquoted string>

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

```

3.5.4. Default 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 a leaf or leaf-list is missing from the configuration and there is a YANG-defined default for that data resource, then the server MUST use the YANG-defined default as the configured value.

If the target of a GET method is a data node that represents a leaf or leaf-list that has a default value, and the leaf or leaf-list has not been instantiated yet, the server MUST return the default value(s) that are in use by the server. In this case, the server MUST ignore its basic-mode, described in [Section 4.8.9](#), and return the default value.

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 an RPC operation defined with the YANG "rpc" statement or a data-model-specific action defined with a YANG "action" statement. It is invoked using a POST method on the operation resource.

An RPC operation is invoked as:

```
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" rpc operation, then invoking the operation would be requested as follows:

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

An action is invoked as:

```
POST {+restconf}/data/<data-resource-identifier>/<action>
```

where <data-resource-identifier> contains the path to the data node where the action is defined, and <action> is the name of the action.

For example, if "module-A" defined a "reset-all" action in the container "interfaces", then invoking this action would be requested as follows:

```
POST /restconf/data/module-A:interfaces/reset-all HTTP/1.1
Server: example.com
```

If the RPC operation is invoked without errors, and if the "rpc" or "action" statement has no "output" section, the response message MUST NOT include a message-body, and MUST send a "204 No Content" status-line instead.

All operation resources representing RPC operations supported by the server MUST be identified in the {+restconf}/operations subtree defined in [Section 3.3.2](#). Operation resources representing YANG actions are not identified in this subtree since they are invoked using a URI within the {+restconf}/data subtree.

3.6.1. Encoding Operation Resource Input Parameters

If the "rpc" or "action" statement has an "input" section then instances of these input parameters are encoded in the module namespace where the "rpc" or "action" statement is defined, in an XML element or JSON object named "input", which is in the module namespace where the "rpc" or "action" statement is defined.

If the "rpc" or "action" statement has an "input" section and the "input" object tree contains any child data nodes which are considered mandatory nodes, then a message-body **MUST** be sent by the client in the request.

If the "rpc" or "action" statement has an "input" section and the "input" object tree does not contain any child nodes which are considered mandatory nodes, then a message-body **MAY** be sent by the client in the request.

If the "rpc" or "action" statement has no "input" section, the request message **MUST NOT** include a message-body.

Examples:

The following YANG module is used for the RPC operation examples in this section.


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

  organization "Example, Inc.";
  contact "support at example.com";
  description "Example Operations Data Model Module";
  revision "2016-07-07" {
    description "Initial version.";
    reference "example.com document 3-3373";
  }

  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 following YANG module is used for the YANG action examples in this section.


```
module example-actions {
  yang-version 1.1;
  namespace "https://example.com/ns/example-actions";
  prefix "act";
  import ietf-yang-types { prefix yang; }

  organization "Example, Inc.";
  contact "support at example.com";
  description "Example Actions Data Model Module";
  revision "2016-07-07" {
    description "Initial version.";
    reference "example.com document 2-9973";
  }

  revision "2016-03-10";

  container interfaces {
    list interface {
      key name;
      leaf name { type string; }

      action reset {
        input {
          leaf delay {
            units seconds;
            type uint32;
            default 0;
          }
        }
      }

      action get-last-reset-time {
        output {
          leaf last-reset {
            type yang:date-and-time;
            mandatory true;
          }
        }
      }
    }
  }
}
```

RPC Input Example:

The client might send the following POST request message to invoke the "reboot" RPC operation:

```
POST /restconf/operations/example-ops:reboot HTTP/1.1
Host: example.com
Content-Type: application/yang-data+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 2016 11:01:00 GMT
Server: example-server
```

The same example request message is shown here using JSON encoding:

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

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

Action Input Example:

The client might send the following POST request message to invoke the "reset" action:

```
POST /restconf/data/example-actions:interfaces/\
  interface=eth0/reset HTTP/1.1
Host: example.com
Content-Type: application/yang-data+xml

<input xmlns="https://example.com/ns/example-actions">
  <delay>600</delay>
</input>
```

The server might respond:


```
HTTP/1.1 204 No Content
Date: Mon, 25 Apr 2016 11:01:00 GMT
Server: example-server
```

The same example request message is shown here using JSON encoding:

```
POST /restconf/data/example-actions:interfaces/\
  interface=eth0/reset HTTP/1.1
Host: example.com
Content-Type: application/yang-data+json

{ "example-actions:input" : {
  "delay" : 600
}
}
```

3.6.2. Encoding Operation Resource Output Parameters

If the "rpc" or "action" statement has an "output" section then instances of these output parameters are encoded in the module namespace where the "rpc" or "action" statement is defined, in an XML element or JSON object named "output", which is in the module namespace where the "rpc" or "action" statement is defined.

If the RPC operation is invoked without errors, and if the "rpc" or "action" statement has an "output" section and the "output" object tree contains any child data nodes which are considered mandatory nodes, then a response message-body **MUST** be sent by the server in the response.

If the RPC operation is invoked without errors, and if the "rpc" or "action" statement has an "output" section and the "output" object tree does not contain any child nodes which are considered mandatory nodes, then a response message-body **MAY** be sent by the server in the response.

The request URI is not returned in the response. Knowledge of the request URI may be needed to associate the output with the specific "rpc" or "action" statement used in the request.

Examples:

RPC Output Example:

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

The client might send the following POST request message to invoke the "get-reboot-info" operation:

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

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 25 Apr 2016 11:10:30 GMT
Server: example-server
Content-Type: application/yang-data+json
```

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

The same response is shown here using XML encoding:

```
HTTP/1.1 200 OK
Date: Mon, 25 Apr 2016 11:10:30 GMT
Server: example-server
Content-Type: application/yang-data+xml

<output xmlns="https://example.com/ns/example-ops">
  <reboot-time>30</reboot-time>
  <message>Going down for system maintenance</message>
  <language>en-US</language>
</output>
```

Action Output Example:

The "example-actions" YANG module defined in [Section 3.6.1](#) is used for this example.

The client might send the following POST request message to invoke the "get-last-reset-time" action:

```
POST /restconf/data/example-actions:interfaces/\
  interface=eth0/get-last-reset-time HTTP/1.1
Host: example.com
Accept: application/yang-data+json
```


The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 25 Apr 2016 11:10:30 GMT
Server: example-server
Content-Type: application/yang-data+json
```

```
{
  "example-actions:output" : {
    "last-reset" : "2015-10-10T02:14:11Z"
  }
}
```

3.6.3. Encoding Operation Resource Errors

If any errors occur while attempting to invoke the operation or action, then an "errors" media type is returned with the appropriate error status.

If the RPC operation input is not valid, or the RPC operation is invoked but errors occur, then a message-body MUST be sent by the server, containing an "errors" resource, as defined in [Section 3.9](#). A detailed example of an operation resource error response can be found in [Section 3.6.3](#).

Using the "reboot" RPC operation from the example in [Section 3.6.1](#), 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-data+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 2016 11:10:30 GMT
Server: example-server
Content-Type: application/yang-data+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:ops="https://example.com/ns/example-ops">
      /ops:input/ops:delay
    </error-path>
    <error-message>Invalid input parameter</error-message>
  </error>
</errors>
```

The same response is shown here in JSON encoding:

```
HTTP/1.1 400 Bad Request
Date: Mon, 25 Apr 2016 11:10:30 GMT
Server: example-server
Content-Type: application/yang-data+json
```

```
{ "ietf-restconf:errors" : {
  "error" : [
    {
      "error-type" : "protocol",
      "error-tag" : "invalid-value",
      "error-path" : "/example-ops:input/delay",
      "error-message" : "Invalid input parameter",
    }
  ]
}
```

3.7. Schema Resource

The server can optionally support retrieval of the YANG modules it supports. If retrieval is supported, then the "schema" leaf MUST be present in the associated "module" list entry, defined in [\[RFC7895\]](#).

To retrieve a YANG module, a client first needs to get the URL for retrieving the schema, which is stored in the "schema" leaf. Note that there is no required structure for this URL. The URL value shown below is just an example.

The client might send the following GET request message:

```
GET /restconf/data/ietf-yang-library:modules-state/\
  module=example-jukebox,2016-08-15/schema HTTP/1.1
Host: example.com
Accept: application/yang-data+json
```


The server might respond:

```
HTTP/1.1 200 OK
Date: Thu, 11 Feb 2016 11:10:30 GMT
Server: example-server
Content-Type: application/yang-data+json

{
  "ietf-yang-library:schema" :
    "https://example.com/mymodules/example-jukebox/2016-08-15"
}
```

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/\
    2016-08-15 HTTP/1.1
Host: example.com
Accept: application/yang
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Thu, 11 Feb 2016 11:10:31 GMT
Server: example-server
Content-Type: application/yang

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](#).

An event 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 the stream resources.

3.9. Errors YANG Data Template

The "errors" YANG data template models 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 as a resource type because no instances can be retrieved with a GET request.

The "ietf-restconf" YANG module contains the "yang-errors" YANG data template, that specifies the syntax and semantics of an "errors" container within a RESTCONF response. RESTCONF error handling behavior is defined in [Section 7](#).

4. RESTCONF Methods

The RESTCONF protocol uses HTTP methods to identify the CRUD operations requested for a particular resource.

The following table shows how the RESTCONF operations relate to NETCONF protocol operations and for the NETCONF <edit-config> operation, the "nc:operation" attribute.

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

CRUD Methods in RESTCONF

The "remove" edit operation attribute for the NETCONF <edit-config> RPC operation 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" edit operation when using a plain patch (see [Section 4.6.1](#)); other media-types may provide more granular control.

Access control mechanisms are used to limit what CRUD 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. 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 RESTCONF operation for any resources that the client is not authorized to access.

Implementation of all methods (except PATCH [[RFC5789](#)]) 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 MUST implement this method.

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

[4.2.](#) HEAD

The RESTCONF server MUST support the HEAD method. The HEAD method is sent by the client to retrieve just the header fields (which contain the metadata for a resource) that would be returned for the comparable GET method, without the response message-body. It is supported for all resources that support the GET method.

The request MUST contain a request URI that contains at least the root resource. 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 RESTCONF server MUST support the GET method. The GET method is sent by the client to retrieve data and metadata 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 root resource.

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 "401 Unauthorized" status-line SHOULD be returned. The error-tag value "access-denied" is returned in this case. A server MAY return a "404 Not Found" status-line, as described in [section 6.5.3 in \[RFC7231\]](#). The error-tag value "invalid-value" is returned in this case.

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.

If any content is returned to the client, then the server MUST send a valid response message-body. More than one element MUST NOT be returned for XML encoding. If multiple elements are sent in a JSON message-body, then they MUST be sent as a JSON array. In this case any timestamp or entity-tag returned in the response MUST be associated with the first element returned.

If a retrieval request for a data resource representing a YANG leaf-list or list object identifies more than one instance, and XML encoding is used in the response, then an error response containing a "400 Bad Request" status-line MUST be returned by the server. The error-tag value "invalid-value" is used in this case. Note that a non-configuration list is not required to defined any keys. In this case, retrieval of a single list instance is not possible.

If a retrieval request for a data resource represents an instance that does not exist, then an error response containing a "404 Not Found" status-line MUST be returned by the server. The error-tag value "invalid-value" is used in this case.

If the target resource of a retrieval request is for an operation resource then a "405 Method Not Allowed" status-line MUST be returned by the server. The error-tag value "operation-not-supported" is used in this case.

Note that the way that access control is applied to data resources may not be completely compatible with HTTP caching. The Last-Modified and ETag header fields maintained for a data resource are not affected by changes to the access control rules for that data resource. It is possible for the representation of a data resource that is visible to a particular client to be changed without detection via the Last-Modified or ETag values.

Example:

The client might request the response header fields for an XML representation of the a specific "album" resource:


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

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2016 17:02:40 GMT
Server: example-server
Content-Type: application/yang-data+xml
Cache-Control: no-cache
ETag: "a74eefc993a2b"
Last-Modified: Mon, 23 Apr 2016 11:02:14 GMT
```

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

Refer to [Appendix D.1](#) for more resource retrieval examples.

4.4. POST

The RESTCONF server MUST support the POST method. The POST method is sent by the client to create a data resource or invoke an operation resource. The server uses the target resource 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 an RPC 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 message-body MUST contain exactly one instance of the

expected data resource. The data-model for the child tree is the subtree as defined by YANG for the child resource.

The "insert" [Section 4.8.5](#) and "point" [Section 4.8.6](#) query parameters MUST be supported by the POST method for datastore and data resources. These parameters are only allowed if the list or leaf-list is ordered-by user.

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

If the data resource already exists, then the POST request MUST fail and a "409 Conflict" status-line MUST be returned. The error-tag value "resource-denied" is used in this case.

If the user is not authorized to create the target resource, an error response containing a "403 Forbidden" status-line SHOULD be returned. The error-tag value "access-denied" is used in this case. A server MAY return a "404 Not Found" status-line, as described in [section 6.5.3 in \[RFC7231\]](#). The error-tag value "invalid-value" is used in this case. 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" : {} }
```

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

```
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2016 17:01:00 GMT
Server: example-server
Location: https://example.com/restconf/data/\
  example-jukebox:jukebox
Last-Modified: Mon, 23 Apr 2016 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" status-line is returned to the client. The error-tag value "access-denied" is used in this case. 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-data+json

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

The server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2016 17:50:00 GMT
Server: example-server
```

4.5. PUT

The RESTCONF server MUST support the PUT method. The PUT method is sent by the client to create or replace the target data resource. A request message-body MUST be present, representing the new data resource, or the server MUST return "400 Bad Request" status-line. The error-tag value "invalid-value" is used in this case.

Both the POST and PUT methods can be used to create data resources. The difference is that for POST, the client does not provide the resource identifier for the resource that will be created. The target resource for the POST method for resource creation is the parent of the new resource. The target resource for the PUT method for resource creation is the new resource.

The PUT method MUST be supported for data and datastore resources. A PUT on the datastore resource is used to replace the entire contents of the datastore. A PUT on a data resource only replaces that data resource within the datastore.

The "insert" ([Section 4.8.5](#)) and "point" ([Section 4.8.6](#)) query parameters MUST be supported by the PUT method for data resources. These parameters are only allowed if the list or leaf-list is ordered-by user.

Consistent with [[RFC7231](#)], if the PUT request creates a new resource, a "201 Created" status-line is returned. If an existing resource is modified, a "204 No Content" status-line is returned.

If the user is not authorized to create or replace the target resource an error response containing a "403 Forbidden" status-line SHOULD be returned. The error-tag value "access-denied" is used in this case.

A server MAY return a "404 Not Found" status-line, as described in [section 6.5.3](#) in [^RFC7231^](#). The error-tag value "invalid-value" is used in this case. All other error responses are handled according to the procedures defined in [^error-reporting^](#).

If the target resource represents a YANG leaf-list, then the PUT method MUST NOT change the value of the leaf-list instance.

If the target resource represents a YANG list instance, then the key leaf values in message-body representation MUST be the same as the key leaf values in the request URI. The PUT method MUST NOT be used to change the key leaf values for a data resource instance.

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:


```
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 2016 17:04:00 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2016 17:04:00 GMT
ETag: "b27480aeda4c"
```

The same request is shown here using XML encoding:

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

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

4.6. PATCH

The RESTCONF server MUST support the PATCH method. 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 mechanism 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, and examining the Accept-Patch header field in the response. (see [Section 4.1](#)).

This document defines one patch mechanism ([Section 4.6.1](#)). Another patch mechanism, the YANG PATCH mechanism, is defined in [\[I-D.ietf-netconf-yang-patch\]](#). Other patch mechanisms may be defined by future specifications.

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" status-line SHOULD be returned. A server MAY return a "404 Not Found" status-line, as described in [section 6.5.3 in \[RFC7231\]](#). The error-tag value "invalid-value" is used in this case. 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. The message-body for a plain patch MUST be present and MUST be represented by the media type "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 the ability to delete child resources. The YANG Patch Media Type allows multiple sub-operations (e.g., merge, delete) within a single PATCH method.

If the target resource represents a YANG leaf-list, then the PATCH method MUST NOT change the value of the leaf-list instance.

If the target resource represents a YANG list instance, then the key leaf values in message-body representation MUST be the same as the key leaf values in the request URI. The PATCH method MUST NOT be used to change the key leaf values for a data resource instance.

After the plain patch is processed by the server. a response will be returned to the client, as specified in [Section 4.6](#).

Example:

To replace just the "year" field in the "album" resource (instead of replacing the entire resource with the PUT method), the client might send a plain patch as follows.

```
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 2016 17:49:30 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2016 17:49:30 GMT
ETag: "b2788923da4c"
```

[4.7.](#) DELETE

The RESTCONF server MUST support the DELETE method. The DELETE method is used to delete the target resource. If the DELETE request succeeds, a "204 No Content" status-line is returned.

If the user is not authorized to delete the target resource then an error response containing a "403 Forbidden" status-line SHOULD be returned. The error-tag value "access-denied" is returned in this case. A server MAY return a "404 Not Found" status-line, as described in [section 6.5.3 in \[RFC7231\]](#). The error-tag value "invalid-value" is returned in this case. All other error responses are handled according to the procedures defined in [Section 7](#).

If the target resource represents a configuration leaf-list or list data node, then it MUST represent a single YANG leaf-list or list instance. The server MUST NOT use the DELETE method to delete more than one such instance.

Example:

To delete the "album" resource with the key "Wasting Light", 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 2016 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.

- o Query parameters can be given in any order.
- o Each parameter can appear at most once in a request URI.
- o If more than one instance of a query parameter is present, then a "400 Bad Request" status-line MUST be returned by the server. The error-tag value "invalid-value" is returned in this case.
- o A default value may apply if the parameter is missing.
- o Query parameter names and values are case-sensitive
- o A server MUST return an error with a '400 Bad Request' status-line if a query parameter is unexpected. The error-tag value "invalid-value" is returned in this case.

Name	Methods	Description
content	GET, HEAD	Select config and/or non-config data resources
depth	GET, HEAD	Request limited sub-tree depth in the reply content
fields	GET, HEAD	Request a subset of the target resource contents
filter	GET, HEAD	Boolean notification filter for event stream resources
insert	POST, PUT	Insertion mode for ordered-by user data resources
point	POST, PUT	Insertion point for ordered-by user data resources
start-time	GET, HEAD	Replay buffer start time for event stream resources
stop-time	GET, HEAD	Replay buffer stop time for event stream resources
with-defaults	GET, HEAD	Control retrieval of default values

RESTCONF Query Parameters

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" status-line is returned if used for other methods or resource types.

If this query parameter is not present, the default value is "all". This query parameter MUST be supported by the server.

4.8.2. The "depth" Query Parameter

The "depth" parameter is used to limit the depth of subtrees returned by the server. Data nodes with a depth value greater than the "depth" parameter are not returned in a response for a GET method.

The requested data node has a depth level of '1'. If the "fields" parameter ([Section 4.8.3](#)) is used to select descendant data nodes, then these nodes and all their ancestor nodes have a depth value of 1. (This has the effect of including the nodes specified by the fields, even if the "depth" value is less than the actual depth level of the specified fields.) Any other child node has 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" status-line 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. The exception is the datastore resource. If this resource type is retrieved then by default the datastore and all child data resources are 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.

The server will return a message-body representing the target resource, with descendant nodes pruned as specified in the

"fields-expr" value. The server does not return a set of separate sub-resources.

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 3.5.3.1](#).

";" 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. Note that there is no path separator character '/' between a "path" field and left parenthesis character '('.

For example, assume the target resource is the "album" list. To retrieve only the "label" and "catalogue-number" of the "admin" container within 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" status-line 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 "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 an event stream resource. A "400 Bad Request" status-line 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, plus any functions defined by the data model.
- 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.

4.8.5. The "insert" Query Parameter

The "insert" parameter is used to specify how a resource should be inserted within a ordered-by user 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 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" status-line is returned.

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

4.8.6. 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 an ordered-by user 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 user.

If the "insert" query parameter is not present, or has a value other than "before" or "after", then a "400 Bad Request" status-line 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.

4.8.7. The "start-time" Query Parameter

The "start-time" parameter is used to trigger the notification replay feature defined in [RFC5277] 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 a "400 Bad Request" status-line.

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" status-line 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. A client can obtain a server's current time by examining the "Date" header field that the server returns in response messages, according to [[RFC7231](#)].

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](#), and the "stop-time" query parameter MUST also be supported by the server.

If the "replay-support" leaf has the value 'true' 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" status-line 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](#), and 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 to the YANG default by the client are 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 \[RFC6243\]](#). Metadata is reported by the server as specified in [Section 5.3](#). The XML encoding for the "default" attribute sent by the server for default nodes is defined in [section 6 of \[RFC6243\]](#). The JSON encoding for the "default" attribute MUST use the same values as defined in [\[RFC6243\]](#), but encoded according to the rules in [\[RFC7952\]](#). The module name "ietf-netconf-with-defaults" MUST be used for the "default" attribute.

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.

Since the server does not report the "also-supported" parameter as described in [section 4.3 of \[RFC6243\]](#), it is possible that some values for the "with-defaults" parameter will not be supported. If the server does not support the requested value of the "with-defaults" parameter, the server MUST return a response with a "400 Bad Request" status-line. The error-tag value "invalid-value" is used in this case.

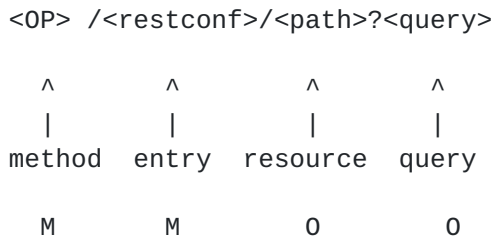
5. Messages

The RESTCONF protocol uses HTTP 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:



M=mandatory, O=optional

where:

- <OP> is the HTTP method
- <restconf> is the RESTCONF root resource
- <path> is the Target Resource URI
- <query> is the query parameter list

- 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 RESTCONF operation. If this field is not present, then the target resource is the API itself, represented by the YANG data template named `"yang-api"`, found in [Section 8](#).
- o query: the set of parameters associated with the RESTCONF message, as defined in [section 3.4 of \[RFC3986\]](#). RESTCONF parameters have the familiar form of `"name=value"` pairs. Most query parameters are optional to implement by the server and optional to use by the client. Each optional query parameter is identified by a URI. The server MUST list the optional 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 [\[RFC3986\], Section 3.4](#). Any reserved characters MUST be percent-encoded, according to [\[RFC3986\], section 2.1](#) and 2.5.

Note that the fragment component not used by the RESTCONF protocol. The fragment is excluded from the target URI by a server, as described in [section 5.1 of \[RFC7230\]](#).

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

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

Refer to [Appendix D](#) for examples of RESTCONF Request URIs.

5.2. 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 one of either XML or JSON encoding. A server MAY support both XML and JSON encoding. XML encoding rules for data nodes are defined in [\[RFC7950\]](#). The same encoding rules are used for all XML

content. JSON encoding rules are defined in [[RFC7951](#)]. Additional JSON encoding rules for metadata are defined in [[RFC7952](#)]. 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 field. This field MUST be present if a message-body is sent by the client.

The server MUST support the "Accept" header field and "406 Not Acceptable" status-line, as defined in [[RFC7231](#)]. The response output content encoding formats that the client will accept are identified with the Accept header field in the request. If it is not specified, the request input encoding format SHOULD be used, or the server MAY choose any supported content encoding format.

If there was no request input, then the default output encoding is XML or JSON, depending on server preference. File extensions encoded in the request are not used to identify format encoding.

A client can determine if the RESTCONF server supports an encoding format by sending a request using a specific format in the Content-Type and/or Accept header field. If the server does not support the requested input encoding for a request, then it MUST return an error response with a '415 Unsupported Media Type' status-line. If the server does not support any of the requested output encodings for a request, then it MUST return an error response with a '406 Not Acceptable' status-line.

[5.3.](#) RESTCONF Metadata

The RESTCONF protocol needs support retrieval of the same metadata 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.

With the XML encoding, the metadata is encoded as attributes in XML, according to section 3.3 of [[W3C.REC-xml-20081126](#)]. With the JSON encoding, the metadata is encoded as specified in [[RFC7952](#)].

The following examples are based on the example in [Appendix D.3.9](#). The "report-all-tagged" mode for the "with-defaults" query parameter requires that a "default" attribute be returned for default nodes. This example shows that attribute for the "mtu" leaf .

[5.3.1.](#) XML Metadata Encoding Example

```
GET /restconf/data/interfaces/interface=eth1
    ?with-defaults=report-all-tagged HTTP/1.1
Host: example.com
Accept: application/yang-data+xml
```

The server might respond as follows.

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2016 17:01:00 GMT
Server: example-server
Content-Type: application/yang-data+xml
```

```
<interface
  xmlns="urn:example.com:params:xml:ns:yang:example-interface">
  <name>eth1</name>
  <mtu xmlns:wd="urn:ietf:params:xml:ns:netconf:default:1.0"
    wd:default="true">1500</mtu>
  <status>up</status>
</interface>
```

[5.3.2.](#) JSON Metadata Encoding Example

Note that [RFC 6243](#) defines the "default" attribute with XSD, not YANG, so the YANG module name has to be assigned instead of derived from the YANG module. The value "ietf-netconf-with-defaults" is assigned for JSON metadata encoding.

```
GET /restconf/data/interfaces/interface=eth1\
    ?with-defaults=report-all-tagged 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 2016 17:01:00 GMT
Server: example-server
Content-Type: application/yang-data+json
```

```
{
  "example:interface" : [
    {
      "name" : "eth1",
      "mtu" : 1500,
      "@mtu" : {
        "ietf-netconf-with-defaults:default" : true
      },
      "status" : "up"
    }
  ]
}
```

5.4. Return Status

Each message represents some sort of resource access. An HTTP "status-line" header field is returned for each request. If a "4xx" range status code is returned in the status-line, then the error information SHOULD be returned in the response, according to the format defined in [Section 7.1](#). If a "5xx" range status code is returned in the status-line, then the error information MAY be returned in the response, according to the format defined in [Section 7.1](#). If a 1xx, 2xx, or 3xx range status code is returned in the status-line, then error information MUST NOT be returned in the response, since these ranges do not represent error conditions.

5.5. 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 field in every response that specifies whether the response should be cached.

Instead of relying on HTTP caching, the client SHOULD track the "ETag" and/or "Last-Modified" header fields 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" header fields, 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 header fields, which SHOULD include the

"ETag" and "Last-Modified" header fields, if this metadata is maintained for the target resource.

Note that the way that access control is applied to data resources the values in the Last-Modified and ETag headers maintained for a data resource may not be reliable, as described in [Section 4.3](#).

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.REC-eventsource-20150203](#)] transport strategy.

6.1. Server Support

A RESTCONF server MAY support RESTCONF notifications. Clients may determine if a server supports RESTCONF notifications by using the HTTP method 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" status-line).

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 method 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-data+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.REC-eventsource-20150203](#)] transport strategy.

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

For example:

The client might send the following request:

```
GET /restconf/data/ietf-restconf-monitoring:restconf-state/\
    streams/stream=NETCONF/access=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-data+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 that 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" event stream defined in [section 3.2.3 of \[RFC5277\]](#). The notification messages for this stream are encoded in XML.

The server MAY support additional streams which represent the semantic content of the NETCONF event stream, but using a representation with a different media type.

The server MAY support the "start-time", "stop-time", and "filter" query parameters, defined in [Section 4.8](#). 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 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 \[RFC5277\]](#) using the XML namespace as defined in the XSD as follows:

```
urn:ietf:params:xml:ns:netconf:notification:1.0
```

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 "eventTime" node is defined within the same XML namespace as the "notification" element. It is defined to be within the "ietf-restconf" module namespace for JSON encoding purposes.

The name and namespace of the payload element are determined by the YANG module containing the notification-stmt representing the notification message.

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


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

  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:netconf:notification:1.0">
data:   <eventTime>2013-12-21T00:01:00Z</eventTime>
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:     "eventTime" : "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:

XML:

```
data: <notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0"><eventTime>2013-12-21T00:01:00Z</eventTime><event \
xmlns="http://example.com/event/1.0"><event-class>fault</event-\
class><reportingEntity><card>Ethernet0</card></reporting-entity>\
<severity>major</severity></event></notification>
```

JSON:

```
data: {"ietf-restconf:notification":{"eventTime":"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 field "Last-Event-Id". RESTCONF servers that do send the "id" field SHOULD support the "start-time" query parameter as the preferred means for a client to specify where to restart the event stream.

7. Error Reporting

HTTP status codes are used to report success or failure for RESTCONF operations. The error information that NETCONF error responses contain in the <rpc-error> element is adapted for use in RESTCONF, and an <errors> data tree information is returned for "4xx" and "5xx" class of status codes.

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

error-tag	status code
in-use	409
invalid-value	400, 404 or 406
(request) too-big	413
(response) too-big	400
missing-attribute	400
bad-attribute	400
unknown-attribute	400
bad-element	400
unknown-element	400
unknown-namespace	400
access-denied	401, 403
lock-denied	409
resource-denied	409
rollback-failed	500
data-exists	409
data-missing	409
operation-not-supported	405 or 501
operation-failed	412 or 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 any resource type, and the status code that will be returned is in the "4xx" range (except for status code "403 Forbidden"), then the server SHOULD send a response message-body containing the information described by the "yang-errors" YANG data template within the "ietf-restconf" module, found in [Section 8](#). The Content-Type of this response message MUST be "application/yang-data", plus optionally a structured syntax name suffix.

The client SHOULD specify the desired encoding(s) for response messages by specifying the appropriate media-type(s) in the Accept header. If the client did not specify an Accept header, then the same structured syntax name suffix used in the request message SHOULD be used, or the server MAY choose any supported message encoding format. If there is no request message the server MUST select "application/yang-data+xml" or "application/yang-data+json", depending on server preference. All of the examples in this document, except for the one below, assume that XML encoding will be returned if there is an error.

YANG Tree Diagram for <errors> data:

```
+----- errors
  +----- error*
    +----- error-type      enumeration
    +----- error-tag       string
    +----- error-app-tag?  string
    +----- error-path?    instance-identifier
    +----- error-message?  string
    +----- error-info?
```

The semantics and syntax for RESTCONF error messages are defined with the "yang-errors" YANG data template extension, found 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 field is used to specify the desired encoding for the error message. There would be no response message-body content if this operation was successful.

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

The server might respond:

```
HTTP/1.1 409 Conflict
Date: Mon, 23 Apr 2016 17:11:00 GMT
Server: example-server
Content-Type: application/yang-data+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 2016 17:11:00 GMT
Server: example-server
Content-Type: application/yang-data+xml
```

```
<errors xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <error>
    <error-type>protocol</error-type>
    <error-tag>data-exists</error-tag>
    <error-path
      xmlns:rc="urn:ietf:params:xml:ns:yang:ietf-restconf"
      xmlns:jbox="https://example.com/ns/example-jukebox">\
      /rc:restconf/rc:data/jbox:jukebox
    </error-path>
    <error-message>
      Data already exists, cannot create new resource
    </error-message>
  </error>
</errors>
```

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" URI).

Note that the "ietf-restconf" module does not have any protocol-accessible objects, so no YANG tree diagram is shown.

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

```
<CODE BEGINS> file "ietf-restconf@2016-08-15.yang"

module ietf-restconf {
```



```
yang-version 1.1;
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>

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

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

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

description
  "This module contains conceptual YANG specifications
  for 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) 2016 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-17.txt

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
revision 2016-08-15 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: RESTCONF Protocol.";
}
```

```
extension yang-data {
  argument name {
    yin-element true;
  }
  description
    "This extension is used to specify a YANG data template which
    represents conceptual data defined in YANG. It is
    intended to describe hierarchical data independent of
    protocol context or specific message encoding format.
    Data definition statements within a yang-data extension
    specify the generic syntax for the specific YANG data
    template, whose name is the argument of the yang-data
    extension statement.
```

Note that this extension does not define a media-type. A specification using this extension MUST specify the message encoding rules, including the content media type.

The mandatory 'name' parameter value identifies the YANG data template that is being defined. It contains the template name.

This extension is ignored unless it appears as a top-level statement. It MUST contain data definition statements that result in exactly one container data node definition. An instance of a YANG data template can thus be translated into an XML instance document, whose top-level element corresponds to the top-level container.

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 are constrained when used within a yang-data-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:yang-data yang-errors {
  uses errors;
}

rc:yang-data 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.";
  }

  anydata error-info {
    description
      "This anydata value MUST represent a container with
      zero or more data nodes representing additional
      error information.";
  }
}
}
```

```
grouping restconf {
  description
    "Conceptual grouping representing the RESTCONF
    root resource.";

  container restconf {
    description
      "Conceptual container representing the RESTCONF
      root resource.";

    container data {
      description
        "Container representing the datastore resource.
        Represents the conceptual root of all state data
        and configuration data supported by the server.
        The child nodes of this container can be any data
        resource which are defined as top-level data nodes
        from the YANG modules advertised by the server in
        the ietf-yang-library module.";
    }
  }
}
```

```
container operations {
  description
    "Container for all operation resources.
```

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

For example, the 'system-restart' RPC operation defined in the 'ietf-system' module would be represented as an empty leaf in the 'ietf-system' namespace. This is a conceptual leaf, and will not actually be found in the module:


```

module ietf-system {
  leaf system-reset {
    type empty;
  }
}

```

To invoke the 'system-restart' RPC operation:

```
POST /restconf/operations/ietf-system:system-restart
```

To discover the RPC operations supported by the server:

```
GET /restconf/operations
```

In XML the YANG module namespace identifies the module:

```

<system-restart
  xmlns='urn:ietf:params:xml:ns:yang:ietf-system' />

```

In JSON the YANG module name identifies the module:

```
{ 'ietf-system:system-restart' : [null] }
```

```

";

```

```

}

```

```
leaf yang-library-version {
```

```

  type string {
    pattern '\d{4}-\d{2}-\d{2}';
  }

```

```

  config false;
  mandatory true;
  description

```

```

    "Identifies the revision date of the ietf-yang-library
    module that is implemented by this RESTCONF server.
    Indicates the year, month, and day in YYYY-MM-DD
    numeric format.";

```

```

  }

```

```

}

```

```

}

```

```

}

```

<CODE ENDS>

9. RESTCONF Monitoring

The "ietf-restconf-monitoring" module provides information about the RESTCONF protocol capabilities and event streams available from the server. A RESTCONF server MUST implement the "ietf-restconf-monitoring" module.

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* [encoding]
        +--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 MAY maintain a last-modified timestamp for this container, and return the "Last-Modified" header field when this data node is retrieved with the GET or HEAD methods. Note that the last-modified timestamp for the datastore resource is not affected by changes to this subtree.

The server SHOULD maintain an entity-tag for this container, and return the "ETag" header field when this data node is retrieved with the GET or HEAD methods. Note that the entity-tag for the datastore resource is not affected by changes to this subtree.

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	4.8.4	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 "basic-mode" defaults handling mode that is used by the server for processing default leafs in requests for data resources. This protocol capability URI MUST be supported by the server, and MUST be listed in the "capability" leaf-list in [Section 9.3](#).

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

RESTCONF defaults capability URI

The URI MUST contain a query parameter named "basic-mode" with one of the values listed below:

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

The "basic-mode" definitions are specified in the "With-Defaults Capability for NETCONF" [[RFC6243](#)].

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 streams supported by the server. The server MAY omit this container if no event 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@2016-08-15.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>

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

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

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

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

    Copyright (c) 2016 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-17.txt

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
revision 2016-08-15 {
  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;
  default false;
  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 encoding;
  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 'text/event-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 XML encoding.
```


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: RFCXXXX

11.2. YANG Module Registry

This document registers two URIs as namespaces in the IETF XML registry [[RFC3688](#)]. Following the format in [RFC 3688](#), the following registration is requested:

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:     RFCXXXX
```

```
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:     RFCXXXX
```


11.3. Media Types

11.3.1. Media Type application/yang-data+xml

Type name: application

Subtype name: yang-data

Required parameters: None

Optional parameters: None

Encoding considerations: 8-bit

Each conceptual YANG data node is encoded according to the XML Encoding Rules and Canonical Format for the specific YANG data node type defined in [[RFC7950](#)].

// RFC Ed.: replace 'NN' in Section NN of [RFCXXXX] with the
// section number for Security Considerations
// Replace 'XXXX' in Section NN of [RFCXXXX] with the actual
// RFC number, and remove this note.

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

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

Interoperability considerations: [RFCXXXX] specifies the format of conforming messages and the interpretation thereof.

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

Published specification: RFC XXXX

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

Fragment identifier considerations: Fragment identifiers for this type are not defined. All YANG data nodes are

accessible as resources using the path in the request URI.

Additional information:

Deprecated alias names for this type: N/A
Magic number(s): N/A
File extension(s): .xml
Macintosh file type code(s): "TEXT"

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

Person & email address to contact for further information: See
Authors' Addresses section of [RFCXXXX].

Intended usage: COMMON

Restrictions on usage: N/A

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

Author: See Authors' Addresses section of [RFCXXXX].

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

Provisional registration? (standards tree only): no

11.3.2. Media Type application/yang-data+json

Type name: application

Subtype name: yang-data+json

Required parameters: None

Optional parameters: None

Encoding considerations: 8-bit

Each conceptual YANG data node is encoded according to
[\[RFC7951\]](#). A data annotation is encoded according to
[\[RFC7952\]](#).

// RFC Ed.: replace 'NN' in Section NN of [RFCXXXX] with the
// section number for Security Considerations
// Replace 'XXXX' in Section NN of [RFCXXXX] with the actual
// RFC number, and remove this note.

Security considerations: Security considerations related to the generation and consumption of RESTCONF messages are discussed in Section NN of [RFCXXXX].

Additional security considerations are specific to the semantics of particular YANG data models. Each YANG module is expected to specify security considerations for the YANG data defined in that module.

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

Interoperability considerations: [RFCXXXX] specifies the format of conforming messages and the interpretation thereof.

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

Published specification: RFC XXXX

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

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

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): .json

Macintosh file type code(s): "TEXT"

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

Person & email address to contact for further information: See Authors' Addresses section of [RFCXXXX].

Intended usage: COMMON

Restrictions on usage: N/A

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

Author: See Authors' Addresses section of [RFCXXXX].

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

Provisional registration? (standards tree only): no

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 review policy for this registry is "IETF Review". The registry shall record for each entry:

- o the name of the RESTCONF capability. By convention, this name begins 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

The "ietf-restconf-monitoring" YANG module defined in this memo is designed to be accessed via the NETCONF protocol [[RFC6241](#)]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [[RFC6242](#)]. The NETCONF access control model [[RFC6536](#)] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [[RFC5246](#)]. The RESTCONF protocol uses the NETCONF access control model [[RFC6536](#)], which provides the means to restrict access for particular RESTCONF users to a preconfigured subset of all available RESTCONF protocol operations and content.

This section provides security considerations for the resources defined by the RESTCONF protocol. Security considerations for HTTPS are defined in [[RFC7230](#)]. RESTCONF does not specify which YANG modules a server needs to support, except the "ietf-restconf-monitoring" module. Security considerations for the other modules manipulated by RESTCONF can be found in the documents defining those YANG modules.

This document does not require use of a specific client authentication mechanism or authorization model, but it does require that a client authentication mechanism and authorization model is used whenever a client accesses a protected resource. Client authentication **MUST** be implemented using client certificates or **MUST** be implemented using an HTTP authentication scheme. Client authorization **MAY** be configured using the NETCONF Access Control Model (NACM) [[RFC6536](#)].

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. There are many patterns of attack that have been observed through operational practice with existing management interfaces. It would be wise for implementers to research them, and take them into account when implementing this protocol.

Different environments may well allow different rights prior to and then after authentication. When a RESTCONF operation is not properly authorized, the RESTCONF server **MUST** return a "401 Unauthorized" status-line. 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. Note that it is possible for a client to detect configuration changes in data resources it is not authorized to access by monitoring changes in the ETag and Last-Modified header fields returned by the server for the datastore resource.

A RESTCONF server implementation SHOULD attempt to prevent system disruption due to excessive resource consumption required to fulfill edit requests via the POST, PUT, and PATCH methods. It may be possible to construct an attack on such a RESTCONF server, which attempts to consume all available memory or other resource types.

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.

The authors would like to thank the following people for their excellent technical reviews of this document: Mehmet Ersue, Mahesh Jethanandani, Qin Wu, Joe Clarke, Bert Wijnen, Ladislav Lhotka, Rodney Cummings, Frank Xialiang, Tom Petch, Robert Sparks, Balint Uveges, Randy Presuhn, Sue Hares, Mark Nottingham, Benoit Claise, Dale Worley, and Lionel Morand.

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

14. References

14.1. Normative References

- [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.
- [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.
- [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.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", [RFC 6242](#), DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [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.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", [RFC 7159](#), DOI 10.17487/RFC7159, March 2014, <<http://www.rfc-editor.org/info/rfc7159>>.
- [RFC7230] Fielding, R. 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.
- [RFC7589] Badra, M., Luchuk, A., and J. Schoenwaelder, "Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication", [RFC 7589](#), DOI 10.17487/RFC7589, June 2015, <<http://www.rfc-editor.org/info/rfc7589>>.
- [RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", [RFC 7895](#), June 2016.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", [RFC 7950](#), DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", [RFC 7951](#), DOI 10.17487/RFC7951, August 2016, <<http://www.rfc-editor.org/info/rfc7951>>.
- [RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", [RFC 7952](#), DOI 10.17487/RFC7952, August 2016, <<http://www.rfc-editor.org/info/rfc7952>>.

[W3C.REC-eventsource-20150203]

Hickson, I., "Server-Sent Events", World Wide Web Consortium Recommendation REC-eventsource-20150203, February 2015, <<http://www.w3.org/TR/2015/REC-eventsource-20150203>>.

[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>>.

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-08](#) (work in progress), March 2016.

[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. v16 to v17

- o various clarifications from NETCONF WG mailing list
- o updated YANG 1.1 and YANG/JSON references to RFC numbers
- o fixed notification namespace and eventTime name bug
- o changed media type application/yang-data-xml to application/yang-data+xml

- o update fragment identifier considerations section for application/yang-data+xml
- o clarify HTTP version requirements

A.2. v15 to v16

- o changed media type application/yang-data to application/yang-data+xml
- o changed header to header field
- o added linewrap convention in terminology and applied in many examples
- o clarified DELETE for leaf-list and list
- o clarified URI format for lists without keys or duplicate leaf-lists
- o added 'yang-data extension' term and clarified 'YANG data template' term
- o clarified that the fragment component is not part of the request URI, per HTTP
- o clarified request URI "api-path" syntax
- o clarified many examples

A.3. v14 to v15

- o added text for HTTP/2 usage
- o changed media type definitions per review comments
- o added some clarifications and typos
- o added error-tag mapping for 406 and 412 errors
- o added clarifications based on ops-dir review by Lionel Morand
- o clarified PUT and POST differences for creating a data resource
- o clarify PUT for a datastore resource
- o added clarifications from Gen-Art review by Robert Sparks

- o clarified terminology in many places

[A.4.](#) v13 - v14

This release addresses github issues #61, #62, #63, #65, #66, and #67.

- o change term 'server' to 'NETCONF server'
- o add term 'RESTCONF server' also called 'server'
- o change term 'client' to 'NETCONF client'
- o add term 'RESTCONF client' also called 'client'
- o remove unused YANG terms
- o clarified operation resource and schema resource terms
- o clarified abstract and intro: RESTCONF uses NETCONF datastore concepts
- o removed term 'protocol operation'; use 'RPC operation' instead
- o clarified edit operation from NETCONF as nc:operation
- o clarified retrieval of an operation resource
- o remove ETag and Last-Modified requirements for /modules-state and /modules-state/module objects, since these are not configuration data nodes
- o clarified Last-Modified and ETag requirements for datastore and data resources
- o clarified defaults retrieval for leaf and leaf-list target resources
- o clarified request message-body for operation resources
- o clarified query parameters for GET also allowed for HEAD
- o clarified error handling for query parameters
- o clarified XPath function library for "filter" parameter
- o added example for 'edit a data resource'

- o added term 'notification replay' from [RFC 5277](#)
- o clarified unsupported encoding format error handling
- o change term 'meta-data' to 'metadata'
- o clarified RESTCONF metadata definition
- o clarified error info not returned for 1xx, 2xx, and 3xx ranges
- o clarified operations description in ietf-restconf module
- o clarified Acknowledgements section
- o clarified some examples
- o update some references
- o update [RFC 2119](#) boilerplate
- o remove requirements that simply restate HTTP requirements
- o remove Pragma: no-cache from examples since [RFC 7234](#) says this pragma is not defined for responses
- o remove suggestion MAY send Pragma: no-cache in response
- o remove table of HTTP status codes used in RESTCONF
- o changed media type names so they conform to [RFC 6838](#)
- o clarified too-big error-tag conversion
- o update SSE reference
- o clarify leaf-list identifier encoding
- o removed all media types except yang-data
- o changed restconf-media-type extension to be more generic yang-data extension

[A.5.](#) v12 - v13

- o fix YANG library module examples (now called module-state)
- o fix terminology idnit issue

- o removed [RFC 2818](#) reference (changed citation to [RFC 7230](#))

[A.6.](#) v11 - v12

- o clarify query parameter requirements
- o move filter query section to match table order in sec. 4.8
- o clarify that depth default is entire subtree for datastore resource
- o change ietf-restconf to YANG 1.1 to use anydata instead of anyxml
- o made implementation of timestamps optional since ETags are mandatory
- o removed confusing text about data resource definition revision date
- o clarify that errors should be returned for any resource type
- o clarified media subtype (not type) for error response
- o clarified client SHOULD (not MAY) specify errors format in Accept header
- o clarified terminology in many sections

[A.7.](#) v10 - v11

- o change term 'operational data' to 'state data'
- o clarify :startup behavior
- o clarify X.509 security text
- o change '403 Forbidden' to '401 Unauthorized' for GET error
- o clarify MUST have one "restconf" link relation
- o clarify that NV-storage is not mandatory
- o clarify how "Last-Modified" and "ETag" header info can be used by a client
- o clarify meaning of mandatory parameter
- o fix module name in action examples

- o clarify operation resource request needs to be known to parse the output
- o clarify ordered-by user terminology
- o fixed JSON example in D.1.1

A.8. v09 - v10

- o address review comments: github issue #36
- o removed intro text about no knowledge of NETCONF needed
- o clarified candidate and confirmed-commit behavior in sec. 1.3
- o clarified that a RESTCONF server MUST support TLS
- o clarified choice of 403 or 404 error
- o fixed forward references to URI template (w/reference at first use)
- o added reference to HTML5
- o made error terminology more consistent
- o clarified that only 1 list or leaf-list instance can be returned in an XML response message-body
- o clarified that more than 1 instance must not be created by a POST method
- o clarified that PUT cannot be used to change a leaf-list value or any list key values
- o clarified that PATCH cannot be used to change a leaf-list value or any list key values
- o clarified that DELETE should not be used to delete more than one instance of a leaf-list or list
- o update JSON RFC reference
- o specified that leaf-list instances are data resources
- o specified how a leaf-list instance identifier is constructed
- o fixed get-schema example

- o clarified that if no Accept header the server SHOULD return the type specified in RESTCONF, but MAY return any media-type, according to HTTP rules
- o clarified that server SHOULD maintain timestamp and etag for data resources
- o clarified default for content query parameter
- o moved terminology section earlier in doc to avoid forward usage
- o clarified intro text wrt/ interactions with NETCONF and access to specific datastores
- o clarified server implementation requirements for YANG defaults
- o clarified that Errors is not a resource, just a media type
- o clarified that HTTP without TLS MUST NOT be used
- o add RESTCONF Extensibility section to make it clear how RESTCONF will be extended in the future
- o add text warning that NACM does not work with HTTP caching
- o remove sec. 5.2 Message Headers
- o remove 202 Accepted from list of used status-lines -- not allowed
- o made implementation of OPTIONS MUST instead of SHOULD
- o clarified that successful PUT for altering data returns 204
- o fixed "point" parameter example
- o added example of alternate value for root resource discovery
- o added YANG action examples
- o fixed some JSON examples
- o changed default value for content query parameter to "all"
- o changed empty container JSON encoding from "[null]" to "{}"
- o added mandatory /restconf/yang-library-version leaf to advertise revision-date of the YANG library implemented by the server

- o clarified URI encoding rules for leaf-list
- o clarified sec. 2.2 wrt/ certificates and TLS
- o added update procedure for entity tag and timestamp

A.9. v08 - v09

- o fix introduction text regarding implementation requirements for the ietf-yang-library
- o clarified HTTP authentication requirements
- o fix host-meta example
- o changed list key encoding to clarify that quoted strings are not allowed. Percent-encoded values are used if quotes would be required. A missing key is treated as a zero-length string
- o Fixed example of percent-encoded string to match YANG model
- o Changed streams examples to align with naming already used

A.10. v07 - v08

- o add support for YANG 1.1 action statement
- o changed mandatory encoding from XML to XML or JSON
- o fix syntax in fields parameter definition
- o add meta-data encoding examples for XML and JSON
- o remove [RFC 2396](#) references and update with 3986
- o change encoding of a key so quoted string are not used, since they are already percent-encoded. A zero-length string is not encoded (/list=foo,,baz)
- o Add example of percent-encoded key value

A.11. v06 - v07

- o fixed all issues identified in email from Jernej Tuljak in netconf email 2015-06-22
- o fixed error example bug where error-urlpath was still used. Changed to error-path.

- o added mention of YANG Patch and informative reference
- o added support for YANG 1.1, specifically support for anydata and actions
- o removed the special field value "*", since it is no longer needed

[A.12.](#) v05 - v06

- o fixed RESTCONF issue #23 (ietf-restconf-monitoring bug)

[A.13.](#) v04 - v05

- 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.14. v03 - v04

- 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.15. v02 - v03

- 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.16.](#) v01 - v02

- 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.17.](#) v00 - v01

- 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.18. 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       instance-identifier
  +--rw player
    +--rw gap? decimal64

```

rpcs:


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

C.1. example-jukebox YANG Module

```
module example-jukebox {

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

  organization "Example, Inc.";
  contact "support at example.com";
  description "Example Jukebox Data Model Module";
  revision "2016-08-15" {
    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 instance-identifier;
    mandatory true;
    description
      "Song identifier. Must identify an instance of
      /jukebox/library/artist/album/song/name.";
  }
}

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

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

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



```
        description "Song number in playlist to play";
    }
}
}
```

[Appendix D](#). RESTCONF Message Examples

The examples within this document use the normative YANG module "ietf-restconf" defined in [Section 8](#) and the non-normative example YANG module "example-jukebox" 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 starts by retrieving the RESTCONF root resource:

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

The server might respond:

```
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>
```

The client may then retrieve the top-level API resource, using the root resource "/restconf".

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

The server might respond as follows:

[RFC Editor Note: Adjust the date (2016-04-09) for ietf-yang-library below to the date in the published ietf-yang-library YANG module, and remove this note.]


```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2016 17:01:00 GMT
Server: example-server
Content-Type: application/yang-data+json
```

```
{
  "ietf-restconf:restconf" : {
    "data" : {},
    "operations" : {},
    "yang-library-version" : "2016-06-21"
  }
}
```

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-data+xml
```

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

[RFC Editor Note: Adjust the date for ietf-yang-library below to the date in the published ietf-yang-library YANG module, and remove this note.]

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2016 17:01:00 GMT
Server: example-server
Cache-Control: no-cache
Content-Type: application/yang-data+xml
```

```
<restconf xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <data/>
  <operations/>
  <yang-library-version>2016-06-21</yang-library-version>
</restconf>
```

[D.1.2.](#) Retrieve The Server Module Information

It is possible the YANG library module will change over time. The client can retrieve the revision date of the ietf-yang-library supported by the server from the API resource, as described in the previous section.

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


```
GET /restconf/data/ietf-yang-library:modules-state HTTP/1.1
Host: example.com
Accept: application/yang-data+json
```

The server might respond as follows:

[RFC Editor Note: Adjust the date for ietf-yang-library below to the date in the published ietf-yang-library YANG module, and remove this note.]

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2016 17:01:00 GMT
Server: example-server
Cache-Control: no-cache
Last-Modified: Sun, 22 Apr 2016 01:00:14 GMT
Content-Type: application/yang-data+json
```

```
{
  "ietf-yang-library:modules-state" : {
    "module-set-id" : "5479120c17a619545ea6aff7aa19838b036ebbd7",
    "module" : [
      {
        "name" : "foo",
        "revision" : "2012-01-02",
        "schema" : "https://example.com/modules/foo/2012-01-02",
        "namespace" : "http://example.com/ns/foo",
        "feature" : [ "feature1", "feature2" ],
        "deviation" : [
          {
            "name" : "foo-dev",
            "revision" : "2012-02-16"
          }
        ],
        "conformance-type" : "implement"
      },
      {
        "name" : "ietf-yang-library",
        "revision" : "2016-06-21",
        "schema" : "https://example.com/modules/\
          ietf-yang-library/2016-06-21",
        "namespace" :
          "urn:ietf:params:xml:ns:yang:ietf-yang-library",
        "conformance-type" : "implement"
      },
      {
        "name" : "foo-types",
        "revision" : "2012-01-05",
        "schema" :
```



```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2016 17:02:00 GMT
Server: example-server
Cache-Control: no-cache
Last-Modified: Sun, 22 Apr 2016 01:00:14 GMT
Content-Type: application/yang-data+xml
```

```
<capabilities
  xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring">
  <capability>\
    urn:ietf:params:restconf:capability:defaults:1.0\
      basic-mode=explicit\
  </capability>
  <capability>\
    urn:ietf:params:restconf:capability:with-defaults: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: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:

```
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2016 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 2016 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:

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

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

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

```
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2016 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 2016 17:03:00 GMT
ETag: "b8389233a4c"
```


D.2.2. Detect Resource Entity-Tag Change

In this example, the server just supports the datastore last-changed timestamp. After the previous request, the client has cached the "Last-Modified" header and the Location header from the response to provide in the following request to patch an "album" list entry with key value "Wasting Light". Only the "genre" field is being updated.

```
PATCH /restconf/data/example-jukebox:jukebox/\
      library/artist=Foo%20Fighters/album=Wasting%20Light/\
      genre HTTP/1.1
Host: example.com
If-Unmodified-Since: Mon, 23 Apr 2016 17:03:00 GMT
Content-Type: application/yang-data+json
```

```
{ "example-jukebox:genre" : "example-jukebox:alternative" }
```

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 2016 19:01:00 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2016 17:45:00 GMT
ETag: "b34aed893a4c"
```

D.2.3. Edit a Datastore Resource

In this example, assume there is a top-level data resource named "system" from the example-system module, and this container has a child leaf called "enable-jukebox-streaming":

```
container system {
  leaf enable-jukebox-streaming {
    type boolean;
  }
}
```

In this example PATCH is used by the client to modify 2 top-level resources at once, in order to enable jukebox streaming and add an "album" sub-resource to each of 2 "artist" resources:

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



```
<data xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <system xmlns="http://example.com/ns/example-system">
    <enable-jukebox-streaming>true</enable-jukebox-streaming>
  </system>
  <jukebox xmlns="http://example.com/ns/example-jukebox">
    <library>
      <artist>
        <name>Foo Fighters</name>
        <album>
          <name>One by One</name>
          <year>2012</year>
        </album>
      </artist>
      <artist>
        <name>Nick Cave and the Bad Seeds</name>
        <album>
          <name>Tender Prey</name>
          <year>1988</year>
        </album>
      </artist>
    </library>
  </jukebox>
</data>
```

[D.2.4.](#) Edit a Data Resource

In this example, the client modifies one data node by adding an "album" sub-resource by sending a PATCH for the data resource:

```
PATCH /restconf/data/example-jukebox:jukebox/library/\
  artist=Nick%20Cave%20and%20the%20Bad%20Seeds HTTP/1.1
Host: example.com
Content-Type: application/yang-data+xml
```

```
<artist xmlns="http://example.com/ns/example-jukebox">
  <name>Nick Cave and the Bad Seeds</name>
  <album>
    <name>The Good Son</name>
    <year>1990</year>
  </album>
</artist>
```

[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", or omitted, since this is the default value. 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 2016 17:11:30 GMT
Server: example-server
Cache-Control: 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". 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 2016 17:11:30 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2016 13:01:20 GMT
ETag: "eeeada438af"
Cache-Control: 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 2016 17:11:30 GMT
Server: example-server
Cache-Control: 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".

```
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 2016 17:11:30 GMT
Server: example-server
Cache-Control: 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" : "/example-jukebox:jukebox/library\
            /artist[name='Foo Figthers']\
            /album[name='Wasting Light']\
            /song[name='Rope']"
        }
      ],
    }
  ]
}
```



```

    {
      "index" : 2,
      "id" : "/example-jukebox:jukebox/library\
        /artist[name='Foo Figthers']\
        /album[name='Wasting Light']\
        /song[name='Bridge Burning']"
    }
  ]
}
],
"player" : {
  "gap" : 0.5
}
}
}

```

Example 2: depth=1

To determine if 1 or more resource instances exist for a given target resource, the value one 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 2016 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Content-Type: application/yang-data+json

```

```

{
  "example-jukebox:jukebox" : {}
}

```

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 2016 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Content-Type: application/yang-data+json
```

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

D.3.3. "fields" Parameter

In this example the client is retrieving the datastore resource in JSON format, but retrieving only the "modules-state/module" list, and only the "name" and "revision" nodes from each list entry. Note that top node returned by the server matches the target resource node (which is "data" in this example). The "module-set-id" leaf is not returned because it is not selected in the fields expression.

```
GET /restconf/data?fields=ietf-yang-library:modules-state/\
    module(name;revision) HTTP/1.1
Host: example.com
Accept: application/yang-data+json
```

The server might respond as follows.

[RFC Editor Note: Adjust the date for ietf-yang-library below to the date in the published ietf-yang-library YANG module, and remove this note.]

[RFC Editor Note: Adjust the date for ietf-restconf-monitoring below to the date in the published ietf-restconf-monitoring YANG module, and remove this note.]


```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2016 17:01:00 GMT
Server: example-server
Content-Type: application/yang-data+json
```

```
{
  "ietf-restconf:data" : {
    "ietf-yang-library:modules-state" : {
      "module" : [
        {
          "name" : "example-jukebox",
          "revision" : "2015-06-04"
        },
        {
          "name" : "ietf-inet-types",
          "revision" : "2013-07-15"
        },
        {
          "name" : "ietf-restconf-monitoring",
          "revision" : "2016-03-16"
        },
        {
          "name" : "ietf-yang-library",
          "revision" : "2016-04-09"
        },
        {
          "name" : "ietf-yang-types",
          "revision" : "2013-07-15"
        }
      ]
    }
  }
}
```

[D.3.4.](#) "insert" Parameter

In this example, a new first song 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[name='Foo Figthers']\
        /album[name='Wasting Light']\
        /song[name='Rope']"
    }
  ]
}
```

Response from server:

```
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2016 13:01:20 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2016 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 entry in the "Foo-One" playlist after the first song.

Request from client:


```
POST /restconf/data/example-jukebox:jukebox/\
  playlist=Foo-One?insert=after&point=\
  %2Fexample-jukebox%3Ajukebox\
  %2Fplaylist%3DFoo-One%2Fsong%3D1 HTTP/1.1
Host: example.com
Content-Type: application/yang-data+json
```

```
{
  "example-jukebox:song" : [
    {
      "index" : 2,
      "id" : "/example-jukebox:jukebox/library\
        /artist[name='Foo Figthers']\
        /album[name='Wasting Light']\
        /song[name='Bridge Burning']"
    }
  ]
}
```

Response from server:

```
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2016 13:01:20 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2016 13:01:20 GMT
Location: https://example.com/restconf/data/\
  example-jukebox:jukebox/playlist=Foo-One/song=2
ETag: "abcada438af"
```

[D.3.6.](#) "filter" Parameter

The following URIs show some examples of notification filter specifications:


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

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

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

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

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

// To get notifications from just two modules (e.g., m1 + m2)
// filter=(/m1:* or /m2:*)
GET /streams/NETCONF?filter=(%2Fm1%3A*%20or%20%2Fm2%3A*)
```

D.3.7. "start-time" Parameter

The following URI shows an example of the "start-time" query parameter:

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

D.3.8. "stop-time" Parameter

The following URI shows an example of the "stop-time" query parameter:

```
// start-time = 2014-10-25T10:02:00Z
// stop-time = 2014-10-25T12:31:00Z
GET /mystreams/NETCONF?start-time=2014-10-25T10%3A02%3A00Z&stop-time=2014-10-25T12%3A31%3A00Z
```


D.3.9. "with-defaults" Parameter

Assume the server implements the module "example" defined in [Appendix A.1 of \[RFC6243\]](#). Assume the server's datastore is 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/example: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 2016 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/example: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 2016 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

