Internet Engineering Task Force (IETF)
Request for Comments: 8040
Category: Standards Track
ISSN: 2070-1721

A. Bierman
YumaWorks
M. Bjorklund
Tail-f Systems
K. Watsen
Juniper Networks
January 2017

RESTCONF Protocol

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 the Network Configuration Protocol (NETCONF).

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc8040.

Table of Contents

1. Introduction

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

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

NETCONF defines configuration datastores and a set of Create, Read, 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 that implements NETCONF datastores.

If a RESTCONF server is co-located with a NETCONF server, then there are protocol interactions with the NETCONF protocol; these interactions 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]:

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

1.1.2. HTTP

The following terms are defined in [RFC3986]:

- fragment
- path
- query

The following terms are defined in [RFC7230]:

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

The following terms are defined in [RFC7231]:

- method
- request
- resource

The following term is defined in [RFC7232]:

- entity-tag

1.1.3. YANG

The following terms are defined in [RFC7950]:

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

1.1.4. NETCONF Notifications

The following term is defined in [RFC5277]:

- notification replay

1.1.5. Terms

The following terms are used within this document:

- 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.

- client: a RESTCONF client.

- data resource: a resource that models a YANG data node. It is defined with YANG data definition statements.

- 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.

- 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").

- event stream resource: a resource that 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 specification [W3C.REC-eventsource-20150203]. Event stream contents are described in Section 3.8.

- 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.

- NETCONF client: a client that implements the NETCONF protocol. Called "client" in [RFC6241].

- NETCONF server: a server that implements the NETCONF protocol. Called "server" in [RFC6241].

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

- operation resource: a resource that models a data-model-specific operation that is in turn defined with a YANG "rpc" or "action" statement. It is invoked with the POST method.

- 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.

- plain patch: a specific media type for use with the PATCH method; see Section 4.6.1. It can be used for simple "merge" edit operations. It is specified by a request Content-Type of "application/yang-data+xml" or "application/yang-data+json".

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

- resource type: one of the RESTCONF resource classes defined in this document. One of "api", "datastore", "data", "operation", "schema", or "event stream".

- RESTCONF capability: an optional RESTCONF protocol feature that is advertised by a particular server if the feature is supported on that server. The feature is identified by an IANA-registered NETCONF Capability URI and advertised with an entry in the "capability" leaf-list defined in Section 9.3.

- RESTCONF client: a client that implements the RESTCONF protocol.

- RESTCONF server: a server that implements the RESTCONF protocol.

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

- schema resource: a resource that is used by the client to retrieve a YANG schema with the GET method. It has a representation with the media type "application/yang".

- server: a RESTCONF server.

- "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.

- stream resource: an event stream resource.

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

- 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 an operational state within the server, so normal YANG data definition statements cannot be used.

- YANG data template: a schema for modeling protocol message components as conceptual data structures 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 A.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 a backslash ("\") as the last character, the line is wrapped for display purposes. It is to be considered to be joined to the next line by deleting the backslash, the following line break, and the leading whitespace of the next line.

1.1.7. Tree Diagrams

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

- Brackets "[" and "]" enclose list keys.
- Abbreviations before data node names: "rw" means configuration data (read-write), "ro" means state data (read-only), and "x" means operation resource (executable).
- Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- Ellipsis ("...") stands for contents of subtrees that are not shown.

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 to 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 HTTP 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 the 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 of 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.

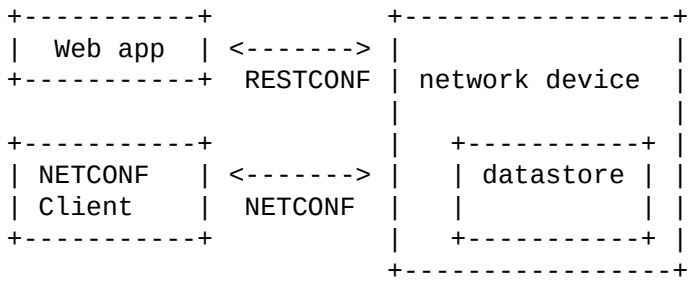
The classification of data as configuration data or non-configuration data is derived from the YANG "config" statement. Behavior related to the ordering of data is derived from the YANG "ordered-by" statement. Non-configuration data is also called "state data".

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.

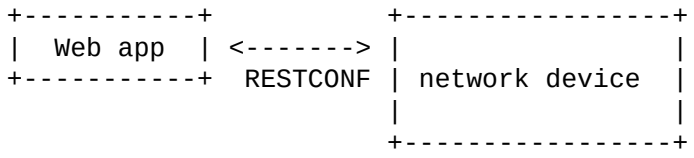
1.4. Coexistence with NETCONF

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:



There are interactions between the NETCONF protocol and RESTCONF protocol related to edit operations. 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 any new 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 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 value "in-use" is returned in this case.

1.5. RESTCONF Extensibility

There are two extensibility mechanisms built into RESTCONF:

- protocol version
- 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 the following:

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

The server might respond as follows:

```
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 "capability" leaf-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 default-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

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 provide both data integrity and confidentiality. A RESTCONF server MUST support the Transport Layer Security (TLS) protocol [RFC5246] and SHOULD adhere to [RFC7525]. 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 (1) use X.509 certificate path validation [RFC5280] to verify the integrity of the RESTCONF server's TLS certificate or (2) 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 3.1 of [RFC2818].

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 a "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 SHOULD require authentication based on TLS client certificates (Section 7.4.6 of [RFC5246]). If certificate-based authentication is not feasible (e.g., because one cannot build the required PKI for clients), then HTTP authentication MAY be used. In the latter case, one of the HTTP authentication schemes defined in the "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry" (Section 5.1 in [RFC7235]) MUST be used.

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 Model (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 the methods allowed on them are specific to the data model.

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 the following:

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

The server might respond as follows:

```
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 the following:

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

The server might respond as follows:

```
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 server. In this example, a custom "play" operation is supported:

The client might send the following:

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

The server might respond as follows:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Cache-Control: no-cache
Last-Modified: Thu, 26 Jan 2017 16:00:14 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 that 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 as follows:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 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. In the example that follows, 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 as follows:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 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-restconf">\
  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" header field (Section 2.2 of [RFC7232]) is returned in the response for a retrieval request. The "If-Unmodified-Since" header field (Section 3.4 of [RFC7232]) 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 co-located with a NETCONF server, then the last-modified timestamp MUST be for the "running" datastore. Note that it is possible that 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" (Section 2.3 of [RFC7232]) 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 co-located with a NETCONF server, then this entity-tag MUST be for the "running" datastore. Note that it is possible that 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 nodes, 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 of its descendants, if any.

A data resource can be retrieved with the GET method. Data resources are accessed via the "{+restconf}/data" URI. This subtree 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 co-located 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 co-located 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 a module other 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:

- The identifier for the leaf-list MUST be encoded using one path segment [RFC3986].
- 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).
- 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 Sections 2.1 and 2.5 of [RFC3986].

- 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.

- The comma (",") character is percent-encoded [RFC3986], 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:

- The key leaf values for a data resource representing a YANG list MUST be encoded using one path segment [RFC3986].

- 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.

- 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.

- 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 Sections 2.1 and 2.5 of [RFC3986]. The comma (",") character MUST be percent-encoded if it is present in the key value.

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

- 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 three key values, and the second key value is a zero-length string.

- Note that non-configuration lists are not required to define keys. In this case, a single list instance cannot be accessed.

- The "list-instance" Augmented Backus-Naur Form (ABNF) [RFC5234] 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" ABNF [RFC5234] 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 "{+restconf}/data" subtree.

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>

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 the 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 or values 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 values 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 that 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 that 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 {
  description "Reboot operation.";
  input {
    leaf delay {
      type uint32;
      units "seconds";
      default 0;
      description
        "Number of seconds to wait before initiating the
         reboot operation.";
    }
    leaf message {
      type string;
      description
        "Log message to display when reboot is started.";
    }
    leaf language {
      type string;
      description "Language identifier string.";
      reference "RFC 5646.";
    }
  }
}
```

```
    rpc get-reboot-info {
      description
        "Retrieve parameters used in the last reboot operation.";
      output {
        leaf reboot-time {
          type uint32;
          description
            "The 'delay' parameter used in the last reboot
             operation.";
        }
        leaf message {
          type string;
          description
            "The 'message' parameter used in the last reboot
             operation.";
        }
        leaf language {
          type string;
          description
            "The 'language' parameter used in the last reboot
             operation.";
        }
      }
    }
  }
```

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.";
    }
```

```
    container interfaces {
      description "System interfaces.";
      list interface {
        key name;
        description "One interface entry.";
        leaf name {
          type string;
          description "Interface name.";
        }

        action reset {
          description "Reset an interface.";
          input {
            leaf delay {
              type uint32;
              units "seconds";
              default 0;
              description
                "Number of seconds to wait before starting the
                 interface reset.";
            }
          }
        }

        action get-last-reset-time {
          description
            "Retrieve the last interface reset time.";
          output {
            leaf last-reset {
              type yang:date-and-time;
              mandatory true;
              description
                "Date and time of the last interface reset, or
                 the last reboot time of the device.";
            }
          }
        }
      }
    }

  }
```

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 as follows:

```
HTTP/1.1 204 No Content
Date: Thu, 26 Jan 2017 20:56:30 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 as follows:

```
HTTP/1.1 204 No Content
Date: Thu, 26 Jan 2017 20:56:30 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 that 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 that 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 as follows:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 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: Thu, 26 Jan 2017 20:56:30 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 as follows:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 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 (1) the RPC operation input is not valid or (2) the RPC operation is invoked but errors occur, then a message-body containing an "errors" resource MUST be sent by the server, as defined in Section 3.9.

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: Thu, 26 Jan 2017 20:56:30 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 using JSON encoding:

```
HTTP/1.1 400 Bad Request
Date: Thu, 26 Jan 2017 20:56:30 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 the retrieval of the YANG modules it uses. 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 as follows:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 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 as follows:

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

   // entire YANG module contents 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 [W3C.REC-eventsource-20150203], using the procedure specified in Section 6.3.

An event stream functions according to the "NETCONF Event 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, which 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.

RESTCONF	NETCONF
OPTIONS	none
HEAD	<get-config>, <get>
GET	<get-config>, <get>
POST	<edit-config> (nc:operation="create")
POST	invoke an RPC operation
PUT	<copy-config> (PUT on datastore)
PUT	<edit-config> (nc:operation="create/replace")
PATCH	<edit-config> (nc:operation depends on PATCH content)
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.

The implementation of all methods (except PATCH [RFC5789]) is 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). The server MUST implement this method.

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 6.5.4 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 define any keys. In this case, the 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 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 as follows:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Content-Type: application/yang-data+xml
Cache-Control: no-cache
ETag: "a74eefc993a2b"
Last-Modified: Thu, 26 Jan 2017 14:02:14 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 B.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 6.5.4 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 the following:

```
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: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Location: https://example.com/restconf/data/\
    example-jukebox:jukebox
Last-Modified: Thu, 26 Jan 2017 20:56:30 GMT
ETag: "b3a3e673be2"
```

Refer to Appendix B.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 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 6.5.4 in [RFC7231]. 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 as follows:

```
HTTP/1.1 204 No Content
Date: Thu, 26 Jan 2017 20:56:30 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 a "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 6.5.4 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.

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 it is created if it does not already exist.

To replace the "album" resource contents, the client might send the following:

```
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 as follows:

```
HTTP/1.1 204 No Content
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Last-Modified: Thu, 26 Jan 2017 20:56:30 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>
```

Refer to Appendix B.2.4 for an example using the PUT method to replace the contents of the datastore resource.

4.6. PATCH

The RESTCONF server MUST support the PATCH method for a plain patch and MAY support additional media types. The media types for the PATCH method supported by the 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).

RESTCONF uses the HTTP PATCH method defined in [RFC5789] to provide an extensible framework for resource patching mechanisms. Each patch mechanism needs a unique media type.

This document defines one patch mechanism (Section 4.6.1). Another patch mechanism, the YANG Patch mechanism, is defined in [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 6.5.4 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 [YANG-Patch] for an alternate media type supporting the ability to delete child resources. The YANG Patch media type allows multiple suboperations (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 as follows:

```
HTTP/1.1 204 No Content
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Last-Modified: Thu, 26 Jan 2017 20:56:30 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 6.5.4 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 the following:

```
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 as follows:

```
HTTP/1.1 204 No Content
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
```

4.8. Query Parameters

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

- Query parameters can be given in any order.
- Each parameter can appear at most once in a request URI.
- 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.
- A default value may apply if the parameter is missing.
- Query parameter names and values are case sensitive.
- 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 subtree 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 the retrieval of default values

RESTCONF Query Parameters

Refer to Appendix B.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" query 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" query 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 of 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 used for other methods or resource types.

By default, the server will include all sub-resources within a retrieved resource that 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 defined 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 a left parenthesis character "(".

For example, assume that 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 defined in Section 9.3, then the server supports the "fields" parameter.

4.8.4. The "filter" Query Parameter

The "filter" query parameter is used to indicate which subset of all possible events is 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 [XPath] and is evaluated in the following context:

- 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.

- The function library is the core function library defined in XPath 1.0, plus any functions defined by the data model.

- The set of variable bindings is empty.

- The context node is the root node.

The "filter" query parameter is used as defined in Section 3.6 of [RFC5277]. 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 defined in Section 9.3, then the server supports the "filter" query parameter.

4.8.5. The "insert" Query Parameter

The "insert" query parameter is used to specify how a resource should be inserted within an "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 "insert" query 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" query 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" query 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 the "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-types" 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 defined 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" query 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-types" 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 defined 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" query 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 described 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 default-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 default-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 default-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 default-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 default-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 defined 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:

```
     <OP> /<restconf>/<path>?<query>

       ^       ^        ^       ^
       |       |        |       |
     method  entry  resource  query

       M       M        O        O
```

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

- 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.

- 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.

- 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.

- query: the set of parameters associated with the RESTCONF message; see 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 "capability" leaf-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 any query parameter value MUST be encoded according to Section 3.4 of [RFC3986]. Any reserved characters MUST be percent-encoded, according to Sections 2.1 and 2.5 of [RFC3986].

Note that the fragment component is 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 B 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. A client will need to support both XML and JSON to interoperate with all RESTCONF servers.

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 it also has special encoding rules to identify module namespaces and provide consistent type processing of YANG data.

The 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 the "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 to support the retrieval of the same metadata that is used in the NETCONF protocol. Information about default leafs, last-modified timestamps, etc. is 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 B.3.9. The "report-all-tagged" mode for the "with-defaults" query parameter requires that a "default" attribute be returned for default nodes. These examples show 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: Thu, 26 Jan 2017 20:56:30 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 the XML Schema Definition (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: Thu, 26 Jan 2017 20:56:30 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 status code in the "4xx" range 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 status code in the "5xx" range 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 status code in the "1xx", "2xx", or "3xx" range 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 MUST 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 access control can be applied to data resources, such that 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 OPTIONS, HEAD, or GET method on the "stream" list. The server does not support RESTCONF notifications if an HTTP error code is returned (e.g., a "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 that 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 B.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;

  organization "Example, Inc.";
  contact "support at example.com";
  description "Example Notification Data Model Module.";
  revision "2016-07-07" {
    description "Initial version.";
    reference "example.com document 2-9976.";
  }

  notification event {
    description "Example notification event.";
    leaf event-class {
      type string;
      description "Event class identifier.";
    }
    container reporting-entity {
      description "Event specific information.";
      leaf card {
        type string;
        description "Line card identifier.";
      }
    }
    leaf severity {
      type string;
      description "Event severity description.";
    }
  }
}
```

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 is whitespace sensitive, the above messages can be encoded onto a single line. For example:

XML:

```
data: <notification xmlns="urn:ietf:params:xml:ns:netconf:notif\
ication: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 specification 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" [W3C.REC-eventsource-20150203]. 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 <errors> data tree information is returned for the "4xx" and "5xx" classes 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 specific to the data model 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 or 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"), 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 a "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 as follows:

```
HTTP/1.1 409 Conflict
Date: Thu, 26 Jan 2017 20:56:30 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 the following:

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

The server might respond as follows:

```
HTTP/1.1 409 Conflict
Date: Thu, 26 Jan 2017 20:56:30 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>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. For example, 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.

<CODE BEGINS>

file "ietf-restconf@2017-01-26.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:   <https://datatracker.ietf.org/wg/netconf/>
     WG List:  <mailto:netconf@ietf.org>

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

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

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

  description
    "This module contains conceptual YANG specifications
     for 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) 2017 IETF Trust and the persons identified as
      authors of the code.  All rights reserved.

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

      This version of this YANG module is part of RFC 8040; see
      the RFC itself for full legal notices.";

    revision 2017-01-26 {
      description
        "Initial revision.";
      reference
        "RFC 8040: RESTCONF Protocol.";
    }

    extension yang-data {
      argument name {
        yin-element true;
      }
      description
        "This extension is used to specify a YANG data template that
         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 values for the YANG module using
        the extension statement are assigned to instance document data
        conforming to the data definition statements within
        this extension.

        The substatements 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 substatements 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 substatements are constrained
        when used within a 'yang-data' 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 are 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 error 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
     resources that 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 the "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 4.8.8	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" default-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 defined 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 "With-defaults Capability for NETCONF" [RFC6243].

If the "basic-mode" is set to "report-all", then the server MUST adhere to the default-handling behavior defined in Section 2.1 of [RFC6243].

If the "basic-mode" is set to "trim", then the server MUST adhere to the default-handling behavior defined in Section 2.2 of [RFC6243].

If the "basic-mode" is set to "explicit", then the server MUST adhere to the default-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.

<CODE BEGINS>

file "ietf-restconf-monitoring@2017-01-26.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:   <https://datatracker.ietf.org/wg/netconf/>
     WG List:  <mailto:netconf@ietf.org>

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

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

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

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

       Copyright (c) 2017 IETF Trust and the persons identified as
       authors of the code.  All rights reserved.

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

       This version of this YANG module is part of RFC 8040; see
       the RFC itself for full legal notices.";

    revision 2017-01-26 {
      description
        "Initial revision.";
      reference
        "RFC 8040: 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 is 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
             subtypes 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.
               The type 'json' is supported for JSON encoding.";
          }
```

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

  }
```

<CODE ENDS>

10. YANG Module Library

The "ietf-yang-library" module defined in [RFC7895] provides information about the YANG modules and submodules used by the RESTCONF server. Implementation is mandatory for RESTCONF servers. All YANG modules and submodules used by the server MUST be identified in the YANG module library.

10.1. modules-state/module

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

The contents of this list are defined in the "module" YANG list statement in [RFC7895].

Note that there are no protocol-accessible objects in the "ietf-restconf" module to implement, but it is possible that a server will list the "ietf-restconf" module in the YANG library if it is imported (directly or indirectly) by an implemented module.

11. IANA Considerations

11.1. The "restconf" Relation Type

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

Relation Name: restconf

Description: Identifies the root of the RESTCONF API as configured on this HTTP server. The "restconf" relation defines the root of the API defined in RFC 8040. Subsequent revisions of RESTCONF will use alternate relation values to support protocol versioning.

Reference: RFC 8040

11.2. Registrations for New URIs and YANG Modules

This document registers two URIs as namespaces in the "IETF XML Registry" [RFC3688]:

```
URI: urn:ietf:params:xml:ns:yang:ietf-restconf
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring
Registrant Contact: The IESG.
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
reference:    RFC 8040

name:         ietf-restconf-monitoring
namespace:    urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring
prefix:       rcmon
reference:    RFC 8040
```

11.3. Media Types

11.3.1. Media Type "application/yang-data+xml"

Type name: application

Subtype name: yang-data+xml

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].

Security considerations: Security considerations related to the generation and consumption of RESTCONF messages are discussed in Section 12 of RFC 8040.
Additional security considerations are specific to the semantics of particular YANG data models. Each YANG module is expected to specify security considerations for the YANG data defined in that module.

Interoperability considerations: RFC 8040 specifies the format of conforming messages and the interpretation thereof.

Published specification: RFC 8040

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): None
Macintosh file type code(s): "TEXT"

Person & email address to contact for further information: See the Authors' Addresses section of RFC 8040.

Intended usage: COMMON

Restrictions on usage: N/A

Author: See the Authors' Addresses section of RFC 8040.

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 metadata annotation is encoded according to
[RFC7952].

Security considerations: Security considerations related
to the generation and consumption of RESTCONF messages
are discussed in Section 12 of RFC 8040.
Additional security considerations are specific to the
semantics of particular YANG data models. Each YANG module
is expected to specify security considerations for the
YANG data defined in that module.

Interoperability considerations: RFC 8040 specifies the format
of conforming messages and the interpretation thereof.

Published specification: RFC 8040

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 the syntax and semantics
specified for the "application/json" media type.

Additional information:

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

Person & email address to contact for further information: See the Authors' Addresses section of RFC 8040.

Intended usage: COMMON

Restrictions on usage: N/A

Author: See the Authors' Addresses section of RFC 8040.

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

Provisional registration? (standards tree only): no

11.4. RESTCONF Capability URNs

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" [RFC5226]. The registry shall record the following for each entry:

- the name of the RESTCONF capability. By convention, this name begins with the colon (":") character.
- the URN for the RESTCONF capability.
- the reference for the document registering the value.

This document registers several capability identifiers in the "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

11.5. Registration of "restconf" URN Sub-namespace

IANA has registered a new URN sub-namespace within the "IETF URN Sub-namespace for Registered Protocol Parameter Identifiers" registry defined in [RFC3553].

Registry Name: restconf

Specification: RFC 8040

Repository: "RESTCONF Capability URNs" registry (Section 11.4)

Index value: Sub-parameters MUST be specified in UTF-8, using standard URI encoding where necessary.

12. Security Considerations

Section 2.1 states that "a RESTCONF server MUST support the TLS protocol [RFC5246]." This language leaves open the possibility that a RESTCONF server might also support future versions of the TLS protocol. Of specific concern, TLS 1.3 [TLS1.3] introduces support for 0-RTT handshakes that can lead to security issues for RESTCONF APIs, as described in Appendix B.1 of the TLS 1.3 document. It is therefore RECOMMENDED that RESTCONF servers do not support 0-RTT at all (not even for idempotent requests) until an update to this RFC guides otherwise.

Section 2.5 recommends authentication based on TLS client certificates but allows the use of any authentication scheme defined in the "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry". Implementations need to be aware that the strengths of these methods vary greatly and that some may be considered experimental. Selection of any of these schemes SHOULD be performed after reading the Security Considerations section of the RFC associated with the scheme's registry entry.

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 preconfigured 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]. Aside from the "ietf-restconf-monitoring" module (Section 9) and the "ietf-yang-library" module (Section 10), RESTCONF does not specify which YANG modules a server needs to support. Security considerations for the other modules manipulated by RESTCONF can be found in the documents defining those YANG modules.

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. On such an implementation, it may be possible to construct an attack that attempts to consume all available memory or other resource types.

13. References

13.1. Normative References

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <http://www.rfc-editor.org/info/rfc2046>.

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

[RFC3553] Mealling, M., Masinter, L., Hardie, T., and G. Klyne, "An IETF URN Sub-namespace for Registered Protocol Parameters", BCP 73, RFC 3553, DOI 10.17487/RFC3553, June 2003, <http://www.rfc-editor.org/info/rfc3553>.

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC5234] Crocker, D., Ed., and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <http://www.rfc-editor.org/info/rfc5277>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <http://www.rfc-editor.org/info/rfc5280>.

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

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

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

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

[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, DOI 10.17487/RFC6243, June 2011, <http://www.rfc-editor.org/info/rfc6243>.

[RFC6415] Hammer-Lahav, E., Ed., and B. Cook, "Web Host Metadata", RFC 6415, DOI 10.17487/RFC6415, October 2011, <http://www.rfc-editor.org/info/rfc6415>.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <http://www.rfc-editor.org/info/rfc6536>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <http://www.rfc-editor.org/info/rfc6570>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <http://www.rfc-editor.org/info/rfc6991>.

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

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

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

[RFC7232] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <http://www.rfc-editor.org/info/rfc7232>.

[RFC7235] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <http://www.rfc-editor.org/info/rfc7235>.

[RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <http://www.rfc-editor.org/info/rfc7320>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/rfc7525>.

[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, DOI 10.17487/RFC7895, June 2016, <http://www.rfc-editor.org/info/rfc7895>.

[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]
Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126>.

[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>.

13.2. Informative References

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

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <http://www.rfc-editor.org/info/rfc2818>.

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

[TLS1.3] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", Work in Progress, draft-ietf-tls-tls13-18, October 2016.

[YANG-Patch]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", Work in Progress, draft-ietf-netconf-yang-patch-14, November 2016.

Appendix A. Example YANG Module

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

YANG tree diagram for the "example-jukebox" module:

```
   +--rw jukebox!
      +--rw library
      |  +--rw artist* [name]
      |  |  +--rw name     string
      |  |  +--rw album* [name]
      |  |     +--rw name     string
      |  |     +--rw genre?   identityref
      |  |     +--rw year?    uint16
      |  |     +--rw admin
      |  |     |  +--rw label?              string
      |  |     |  +--rw catalogue-number?   string
      |  |     +--rw song* [name]
      |  |        +--rw name        string
      |  |        +--rw location    string
      |  |        +--rw format?     string
      |  |        +--rw length?     uint32
      |  +--ro artist-count?   uint32
      |  +--ro album-count?    uint32
      |  +--ro song-count?     uint32
      +--rw playlist* [name]
      |  +--rw name           string
      |  +--rw description?   string
      |  +--rw song* [index]
      |     +--rw index    uint32
      |     +--rw id       instance-identifier
      +--rw player
         +--rw gap?   decimal64

   rpcs:

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

A.1. "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 "artists";
         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
              "The playlist name.";
          }
          leaf song-number {
            type uint32;
            mandatory true;
            description
              "Song number in playlist to play.";
          }
        }
      }
    }
```

Appendix B. RESTCONF Message Examples

The examples within this document use the normative YANG module "ietf-restconf" as defined in Section 8 and the non-normative example YANG module "example-jukebox" as defined in Appendix A.1.

This section shows some typical RESTCONF message exchanges.

B.1. Resource Retrieval Examples

B.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 as follows:

```
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:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 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 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:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 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>
```

B.1.2. Retrieve the Server Module Information

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

In this example, the client is retrieving the module 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:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Cache-Control: no-cache
Last-Modified: Thu, 26 Jan 2017 14:00:14 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" :
          "https://example.com/modules/foo-types/2012-01-05",
        "namespace" : "http://example.com/ns/foo-types",
        "conformance-type" : "import"
      },
```

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

B.1.3. Retrieve the Server Capability Information

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

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

The server might respond as follows:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Cache-Control: no-cache
Last-Modified: Thu, 26 Jan 2017 16:00:14 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>
```

B.2. Data Resource and Datastore Resource Examples

B.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: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Location: https://example.com/restconf/data/\
    example-jukebox:jukebox/library/artist=Foo%20Fighters
Last-Modified: Thu, 26 Jan 2017 20:56:30 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: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Location: https://example.com/restconf/data/\
    example-jukebox:jukebox/library/artist=Foo%20Fighters/\
    album=Wasting%20Light
Last-Modified: Thu, 26 Jan 2017 20:56:30 GMT
ETag: "b8389233a4c"
```

B.2.2. Detect Datastore Resource Entity-Tag Change

In this example, the server just supports the datastore last-changed timestamp. Assume that the client has cached the "Last-Modified" header from the response to the previous request. This value is used as in the "If-Unmodified-Since" header in the following request to patch an "album" list entry with a key value of "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: Thu, 26 Jan 2017 20:56:30 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 as follows:

```
HTTP/1.1 412 Precondition Failed
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Last-Modified: Thu, 26 Jan 2017 19:41:00 GMT
ETag: "b34aed893a4c"
```

B.2.3. Edit a Datastore Resource

In this example, assume that 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 two top-level resources at once, in order to enable jukebox streaming and add an "album" sub-resource to each of two "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>
```

B.2.4. Replace a Datastore Resource

In this example, the entire configuration datastore contents are being replaced. Any child nodes not present in the <data> element but present in the server will be deleted.

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

<data xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <jukebox xmlns="http://example.com/ns/example-jukebox">
    <library>
      <artist>
        <name>Foo Fighters</name>
        <album>
          <name>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>
```

B.2.5. 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>
```

B.3. Query Parameter Examples

B.3.1. "content" Parameter

The "content" parameter is used to select the types of data child resources (configuration and/or non-configuration) that are returned by the server for a GET method request.

In this example, a simple YANG list is used 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 of the child resources, the "content" parameter is set to "all", or omitted, since this is the default value. The client might send the following:

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

The server might respond as follows:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 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 as follows:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Last-Modified: Thu, 26 Jan 2017 16:45:20 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 the following:

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

The server might respond as follows:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 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
      }
    ]
  }
}
```

B.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 the retrieval of the top-level "jukebox" data resource.

Example 1: depth=unbounded

To retrieve all of the child resources, the "depth" parameter is not present or is 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 as follows:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 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 Fighters']\
                   /album[name='Wasting Light']\
                   /song[name='Rope']"
              },
              {
                "index" : 2,
                "id" : "/example-jukebox:jukebox/library\
                   /artist[name='Foo Fighters']\
                   /album[name='Wasting Light']\
                   /song[name='Bridge Burning']"
              }
            ]
          }
        ],
        "player" : {
          "gap" : 0.5
        }
      }
    }
```

Example 2: depth=1

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

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

The server might respond as follows:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 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 two 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 as follows:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 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
    }
  }
}
```

B.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 the 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:

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

   {
     "ietf-restconf:data" : {
       "ietf-yang-library:modules-state" : {
         "module" : [
           {
             "name" : "example-jukebox",
             "revision" : "2016-08-15"
           },
           {
             "name" : "ietf-inet-types",
             "revision" : "2013-07-15"
           },
           {
             "name" : "ietf-restconf-monitoring",
             "revision" : "2017-01-26"
           },
           {
             "name" : "ietf-yang-library",
             "revision" : "2016-06-21"
           },
           {
             "name" : "ietf-yang-types",
             "revision" : "2013-07-15"
           }
         ]
       }
     }
   }
```

B.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 Fighters']\
          /album[name='Wasting Light']\
          /song[name='Rope']"
     }
   ]
}
```

Response from server:

```
HTTP/1.1 201 Created
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Last-Modified: Thu, 26 Jan 2017 20:56:30 GMT
Location: https://example.com/restconf/data/\
    example-jukebox:jukebox/playlist=Foo-One/song=1
ETag: "eeeada438af"
```

B.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 Fighters']\
          /album[name='Wasting Light']\
          /song[name='Bridge Burning']"
     }
   ]
}
```

Response from server:

```
HTTP/1.1 201 Created
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Last-Modified: Thu, 26 Jan 2017 20:56:30 GMT
Location: https://example.com/restconf/data/\
    example-jukebox:jukebox/playlist=Foo-One/song=2
ETag: "abcada438af"
```

B.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 the 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*)
```

B.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
```

B.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
```

B.3.9. "with-defaults" Parameter

Assume that the server implements the module "example" defined in Appendix A.1 of [RFC6243], and assume that the server's datastore is as defined in Appendix A.2 of [RFC6243].

If the server's "basic-mode" parameter in the "defaults" protocol capability URI (Section 9.1.2) is "trim", 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: Thu, 26 Jan 2017 20:56:30 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's default-handling "basic-mode" parameter 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: Thu, 26 Jan 2017 20:56:30 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.

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 S&TCD.

Authors' Addresses

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net