Abstract

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

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 20, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

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

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

This document describes an HTTP [RFC7230] based protocol called RESTCONF, for accessing data defined in YANG version 1 [RFC6020] or YANG version 1.1 [I-D.ietf-netmod-rfc6020bis], using datastores defined in NETCONF [RFC6241].

The NETCONF protocol defines configuration datastores and a set of Create, Retrieve, Update, Delete (CRUD) operations that can be used to access these datastores. The YANG language defines the syntax and semantics of datastore content, operational data, protocol operations, and event notifications. RESTCONF uses HTTP operations to provide CRUD operations on a NETCONF datastore containing YANG-defined data. Since NETCONF protocol operations are not relevant, the user should not need any prior knowledge of NETCONF in order to use RESTCONF.

Configuration data and state data are exposed as resources that can be retrieved with the GET method. Resources representing configuration data can be modified with the DELETE, PATCH, POST, and PUT methods. Data is encoded with either XML [W3C.REC-xml-20081126] or JSON [RFC7158].

Data-model specific protocol operations defined with the YANG "rpc" 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. Simple Subset of NETCONF Functionality

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

The HTTP POST, PUT, PATCH, and DELETE methods are used to edit data resources represented by YANG data models. These basic edit operations allow the running configuration to be altered in an all-or-none fashion. This is similar to the "rollback-on-error" capability in NETCONF. Edits are usually applied to one data resource instance at a time.
The base RESTCONF protocol is intentionally simple to allow deployment for as many use cases as possible. Additional functionality can be defined in external documents, outside the scope of this document.

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

The following figure shows the system components:

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

1.2. Data Model Driven API

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

In contrast, a REST client using HATEOAS principles would not use any data modeling language to define the application-specific content of the API. The client would need to discover each new child resource as it traverses the URIs to discover the server capabilities. This approach has the following significant weaknesses with regards to control of complex networking devices:

- inefficient performance: configuration APIs will be quite complex and may require thousands of protocol messages to discover all the schema information. Typically the data type information has to be passed in the protocol messages, which is also wasteful overhead.
- no data model richness: without a data model, the schema-level semantics and validation constraints are not available to the application.
o no tool automation: API automation tools need some sort of content
schema to function. Such tools can automate various programming
and documentation tasks related to specific data models.

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

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

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

The RESTCONF protocol operates on a conceptual datastore defined with
the YANG data modeling language. The server lists each YANG module
it supports using the "ietf-yang-library" YANG module, defined in
[I-D.ietf-netconf-yang-library]. The server MUST implement the
"ietf-yang-library" module, which SHOULD identify all the YANG
modules used by the server.

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

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

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

1.3. Coexistence with NETCONF

RESTCONF can be implemented on a device that supports NETCONF.

If the device supports :writable-running, all edits to configuration
nodes in (+restconf)/data are performed in the running configuration
datastore.
Otherwise, if the device supports :candidate, all edits to configuration nodes in [+restconf]/data are performed in the candidate configuration datastore. The candidate is automatically committed to running after a successful edit.

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

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

1.4. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

1.4.1. NETCONF

The following terms are defined in [RFC6241]:

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

1.4.2. HTTP
The following terms are defined in [RFC3986]:
  o fragment
  o path
  o query
The following terms are defined in [RFC7230]:
  o header
  o message-body
  o request-line
  o request URI
  o status-line
The following terms are defined in [RFC7231]:
  o method
  o request
  o resource
The following terms are defined in [RFC7232]:
  o entity tag

1.4.3. YANG

The following terms are defined in [I-D.ietf-netmod-rfc6020bis]:
  o action
  o container
  o data node
  o key leaf
  o leaf
  o leaf-list
o list
o non-presence container (or NP-container)
o ordered-by system
o ordered-by user
o presence container (or P-container)
o RPC operation (now called protocol operation)

1.4.4. Terms

The following terms are used within this document:

o API resource: a resource with the media type "application/yang.api+xml" or "application/yang.api+json".

o data resource: a resource with the media type "application/yang.data+xml" or "application/yang.data+json". Containers, leafs, list entries, anydata and anyxml nodes can be data resources.

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

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

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

o media-type: HTTP uses Internet media types [RFC2046] in the Content-Type and Accept header fields in order to provide open and extensible data typing and type negotiation.

o operation: the conceptual RESTCONF operation for a message, derived from the HTTP method, request URI, headers, and message-body.
o operation resource: a resource with the media type "application/yang.operation+xml" or "application/yang.operation+json".

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

o plain patch: a specific PATCH request type that can be used for simple merge operations.

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

o RESTCONF capability: An optional RESTCONF protocol feature supported by the server, which is identified by an IANA registered NETCONF Capability URI, and advertised with an entry in the "capability" leaf-list in Section 9.3.

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

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

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

o stream list: the set of data resource instances that describe the event stream resources available from the server. This information is defined in the "ietf-restconf-monitoring" module as the "stream" list. It can be retrieved using the target resource "+restconf"/data/ietf-restconf-monitoring:restconf-state/streams/stream". The stream list contains information about each stream, such as the URL to retrieve the event stream data.

1.4.5. URI Template

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

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

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

- Brackets "[" and "]" enclose list keys.
- Abbreviations before data node names: "rw" means configuration data (read-write) and "ro" state data (read-only).
- 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.

2. Transport Protocol Requirements

2.1. Integrity and Confidentiality

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

2.2. HTTPS with X.509v3 Certificates

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

2.3. Certificate Validation

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

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

2.5. Authenticated Client Identity

The RESTCONF server MUST authenticate client access to any protected resource using HTTP Authentication [RFC7235]. If the RESTCONF client is not authenticated to access a resource, the server MUST send a response with status code 401 (Unauthorized) and a WWW-Authenticate header field containing at least one challenge applicable to the target resource. The RESTCONF server MAY advertise support for any number of authentication schemes but, in order to ensure interoperability, the RESTCONF server MUST advertise at least one of the following authentication schemes:

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

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

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

3. Resources

The RESTCONF protocol operates on a hierarchy of resources, starting with the top-level API resource itself (Section 3.1). Each resource represents a manageable component within the device.
A resource can be considered a collection of conceptual data and the set of allowed methods on that data. It can contain nested child resources. The child resource types and methods allowed on them are data-model specific.

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

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

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

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

3.1. Root Resource Discovery

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

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

Response
---------
HTTP/1.1 200 OK
Content-Type: application/xrd+xml
Content-Length: nnn
Once discovering the RESTCONF API root, the client MUST prepend it to any subsequent request to a RESTCONF resource. For instance, using the "/restconf" path discovered above, the client can now determine the operations supported by the server. In this example a custom "play" operation is supported:

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

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

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

3.2. RESTCONF Resource Types

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

<table>
<thead>
<tr>
<th>Resource</th>
<th>Media Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>application/yang.api+xml</td>
</tr>
<tr>
<td></td>
<td>application/yang.api+json</td>
</tr>
<tr>
<td>Datastore</td>
<td>application/yang.datastore+xml</td>
</tr>
<tr>
<td></td>
<td>application/yang.datastore+json</td>
</tr>
<tr>
<td>Data</td>
<td>application/yang.data+xml</td>
</tr>
<tr>
<td></td>
<td>application/yang.data+json</td>
</tr>
<tr>
<td>Errors</td>
<td>application/yang.errors+xml</td>
</tr>
<tr>
<td></td>
<td>application/yang.errors+json</td>
</tr>
<tr>
<td>Operation</td>
<td>application/yang.operation+xml</td>
</tr>
<tr>
<td></td>
<td>application/yang.operation+json</td>
</tr>
<tr>
<td>Schema</td>
<td>application/yang</td>
</tr>
</tbody>
</table>
RESTCONF Media Types

3.3. API Resource

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

YANG Tree Diagram for an API Resource:

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

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

The API resource can be retrieved with the GET method.

This resource has the following child resources:

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

RESTCONF API Resource

3.3.1. "+restconf[/data"

This mandatory resource represents the combined configuration and operational data resources that can be accessed by a client. It cannot be created or deleted by the client. The datastore resource type is defined in Section 3.4.

Example:

This example request by the client would retrieve only the non-configuration data nodes that exist within the "library" resource, using the "content" query parameter (see Section 4.8.1).
GET /restconf/data/example-jukebox:jukebox/library
   ?content=nonconfig HTTP/1.1
Host: example.com
Accept: application/yang.data+xml

The server might respond:

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

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

3.3.2. {+restconf}/operations

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

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

Operation resources are defined in Section 3.6.

3.4. Datastore Resource

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

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

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

3.4.1. Edit Collision Detection

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

3.4.1.1. Timestamp

The last change time is maintained and the "Last-Modified" ([RFC7232], Section 2.2) header is returned in the response for a retrieval request. The "If-Unmodified-Since" header can be used in edit operation requests to cause the server to reject the request if the resource has been modified since the specified timestamp.

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

3.4.1.2. Entity tag

A unique opaque string is maintained and the "ETag" ([RFC7232], Section 2.3) header is returned in the response for a retrieval request. The "If-Match" header can be used in edit operation requests to cause the server to reject the request if the resource entity tag does not match the specified value.

The server MUST maintain an entity tag for the top-level (+restconf)/data resource and SHOULD maintain entity tags for descendant resources. For all resources, the server MUST return the "ETag" header when the resource is retrieved with the GET or HEAD methods. If the server does not maintain an entity tag for a resource, it MUST return the entity tag of the resource’s ancestor, a process that may recurse up to the top-level (+restconf)/data resource. Only changes to configuration data resources within the datastore affect the entity tag.

3.5. Data Resource

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

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

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

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

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

A configuration data resource can be altered by the client with some or all of the edit operations, depending on the target resource and the specific operation. Refer to Section 4 for more details on edit operations.

The resource definition version for a data resource is identified by the revision date of the YANG module containing the YANG definition for the data resource.

3.5.1. Encoding Data Resource Identifiers in the Request URI

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

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

A RESTCONF data resource identifier is not an XPath expression. It is encoded from left to right, starting with the top-level data node,
according to the "api-path" rule in Section 3.5.1.1. The node name of each ancestor of the target resource node is encoded in order, ending with the node name for the target resource. If a node in the path is defined in another module than its parent node, then module name followed by a colon character (":") is prepended to the node name in the resource identifier. See Section 3.5.1.1 for details.

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 "=" followed by the single key leaf value.

- If there are multiple key leaf values, the value of each leaf identified in the "key" statement is encoded in the order specified in the YANG "key" statement, with a comma separating them.

- The key value is specified as a string, using the canonical representation for the YANG data type. Any reserved characters MUST be percent-encoded, according to [RFC3986], section 2.1.

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

- Quoted strings are not allowed in the key leaf values. A missing key value is interpreted a zero-length string. (example: list=foo,,baz).

- The "list-instance" ABNF rule defined in Section 3.5.1.1 represents the syntax of a list instance identifier.

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

Examples:

```plaintext
container top {
  list list1 {
    key "key1 key2 key3";
  }
}
```
For the above YANG definition, URI with key leaf values will be encoded as follows (line wrapped for display purposes only):

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

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, space, and forward slash. (',"":/'). Note that the angle brackets ('<', '>'), and double-quote ('"') are not reserved characters and do not need to be percent-encoded.

Example URL:

```
/restconf/data/example-top:top/list2=key1/X

/restconf/data/example-top:top/list2=%2C%27%22%3A%20%2F/X
```

3.5.1.1. ABNF For Data Resource Identifiers

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

```
api-path = "/" | 
  ("/" api-identifier
   0*("/" (api-identifier | list-instance )))
api-identifier = [module-name ":"] identifier ;; note 1
module-name = identifier
list-instance = api-identifier "=" key-value ["," key-value]*
key-value = string ;; note 1
string = <a quoted or unquoted string>
  ;; An identifier MUST NOT start with
  ;; ('X'|'x') ('M'|'m') ('L'|'l'))
```
identifier  = (ALPHA / "_")
    *(ALPHA / DIGIT / "_" / "-" / ".")

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

3.5.2. Defaults Handling

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

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

If the target of a GET method is a data node that represents a container or list that has any child resources with default values, for the child resources that have not been given value yet, the server MAY return the default values that are in use by the server, in accordance with its reported default handing mode and query parameters passed by the client.

3.6. Operation Resource

An operation resource represents a protocol operation defined with one of the YANG "action" or "rpc" statements. 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 from "module-A" 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>/<operation>
where \(<\text{data-resource-identifier}>\) contains the path to the data node where the action is defined, and \(<\text{operation}>\) 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 "action" or "rpc" statement has an "input" section, then a message-body MAY be sent by the client in the request, otherwise the request message MUST NOT include a message-body.

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

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

3.6.1. Encoding Operation Input Parameters

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

Example:

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

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

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

leaf language { type string; }
}

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

The client might send the following POST request message:

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

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

The server might respond:

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

3.6.2. Encoding Operation Output Parameters

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

Example:

The "example-ops" YANG module defined in Section 3.6.1 is used for the examples in this section.

The client might send the following POST request message:
POST /restconf/operations/example-ops:get-reboot-info HTTP/1.1
Host: example.com
Accept: application/yang.operation+json

The server might respond:

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

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

3.6.3. Encoding Operation Errors

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

Using the "reset" operation example above, the client might send the following POST request message:

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

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

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

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

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

3.7. Schema Resource

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

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

The client might send the following GET request message:

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

The server might respond:

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

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

Next the client needs to retrieve the actual YANG schema.

The client might send the following GET request message:

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

The server might respond:
module example-jukebox {
    // contents of YANG module deleted for this example...
}

3.8. Event Stream Resource

An "event stream" resource represents a source for system generated event notifications. Each stream is created and modified by the server only. A client can retrieve a stream resource or initiate a long-poll server sent event stream, using the procedure specified in Section 6.3.

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

3.9. Errors Media Type

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

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

4. Operations

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

The following table shows how the RESTCONF operations relate to NETCONF protocol operations:

<table>
<thead>
<tr>
<th>RESTCONF</th>
<th>NETCONF</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIONS</td>
<td>none</td>
</tr>
<tr>
<td>HEAD</td>
<td>none</td>
</tr>
<tr>
<td>GET</td>
<td>&lt;get-config&gt;, &lt;get&gt;</td>
</tr>
<tr>
<td>POST</td>
<td>&lt;edit-config&gt; (operation=&quot;create&quot;)</td>
</tr>
<tr>
<td>PUT</td>
<td>&lt;edit-config&gt; (operation=&quot;create/replace&quot;)</td>
</tr>
</tbody>
</table>
The NETCONF "remove" operation attribute is not supported by the HTTP DELETE method. The resource must exist or the DELETE method will fail. The PATCH method is equivalent to a "merge" operation when using a plain patch (see Section 4.6.1), other media-types may provide more granular control.

Access control mechanisms may be used to limit what operations can be used. In particular, RESTCONF is compatible with the NETCONF Access Control Model (NACM) [RFC6536], as there is a specific mapping between RESTCONF and NETCONF operations, defined in Table 1. The resource path needs to be converted internally by the server to the corresponding YANG instance-identifier. Using this information, the server can apply the NACM access control rules to RESTCONF messages.

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

Implementation of all methods (except PATCH) are defined in [RFC7231]. This section defines the RESTCONF protocol usage for each HTTP method.

4.1. OPTIONS

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

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

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

4.2. HEAD

The HEAD method is sent by the client to retrieve just the headers that would be returned for the comparable GET method, without the response message-body. It is supported for all resource types, except operation resources.
The request MUST contain a request URI that contains at least the entry point. The same query parameters supported by the GET method are supported by the HEAD method.

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

4.3. GET

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

The server MUST NOT return any data resources for which the user does not have read privileges. If the user is not authorized to read the target resource, an error response containing a "403 Forbidden" or "404 Not Found" status-line is returned to the client.

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

Example:

The client might request the response headers for an XML representation of the "library" resource:

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

The server might respond:

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

<album xmlns="http://example.com/ns/example-jukebox">
    <name>Wasting Light</name>
</album>
4.4.  POST

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datastore</td>
<td>Create a top-level configuration data resource</td>
</tr>
<tr>
<td>Data</td>
<td>Create a configuration data child resource</td>
</tr>
<tr>
<td>Operation</td>
<td>Invoke a protocol operation</td>
</tr>
</tbody>
</table>

Resource Types that Support POST

4.4.1.  Create Resource Mode

If the target resource type is a datastore or data resource, then the POST is treated as a request to create a top-level resource or child resource, respectively. The message-body is expected to contain the content of a child resource to create within the parent (target resource). The data-model for the child tree is the subtree is defined by YANG for the child resource.

The "insert" and "point" query parameters are supported by the POST method for datastore and data resource types, as specified in the YANG definition in Section 8.

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

If the user is not authorized to create the target resource, an error response containing a "403 Forbidden" or "404 Not Found" status-line is returned to the client. All other error responses are handled according to the procedures defined in Section 7.

Example:

To create a new "jukebox" resource, the client might send:
POST /restconf/data HTTP/1.1
Host: example.com
Content-Type: application/yang.data+json

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

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

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

Refer to Appendix D.2.1 for more resource creation examples.

4.4.2. Invoke Operation Mode

If the target resource type is an operation resource, then the POST method is treated as a request to invoke that operation. The message-body (if any) is processed as the operation input parameters. Refer to Section 3.6 for details on operation resources.

If the POST request succeeds, a "200 OK" status-line is returned if there is a response message-body, and a "204 No Content" status-line is returned if there is no response message-body.

If the user is not authorized to invoke the target operation, an error response containing a "403 Forbidden" or "404 Not Found" status-line is returned to the client. All other error responses are handled according to the procedures defined in Section 7.

Example:

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

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

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

{

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

The server might respond:

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

4.5. PUT

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

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

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

Consistent with [RFC7231], if the PUT request creates a new resource, a "201 Created" status-line is returned. If an existing resource is modified, either "200 OK" or "204 No Content" are returned.

If the user is not authorized to create or replace the target resource an error response containing a "403 Forbidden" or "404 Not Found" status-line is returned to the client. All other error responses are handled according to the procedures defined in Section 7.

Example:

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

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

```plaintext
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
```
If the resource is updated, the server might respond:

HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:04:00 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 17:04:00 GMT
ETag: b27480aed4a

4.6. PATCH

RESTCONF uses the HTTP PATCH method defined in [RFC5789] to provide an extensible framework for resource patching mechanisms. It is optional to implement by the server. Each patch type needs a unique media type. Zero or more PATCH media types MAY be supported by the server. The media types supported by a server can be discovered by the client by sending an OPTIONS request (see Section 4.1).

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

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

If the user is not authorized to alter the target resource an error response containing a "403 Forbidden" or "404 Not Found" status-line is returned to the client. All other error responses are handled according to the procedures defined in Section 7.

4.6.1. Plain Patch

The plain patch mechanism merges the contents of the message body with the target resource. If the target resource is a datastore resource (see Section 3.4), the message body MUST be either application/yang.datastore+xml or application/yang.datastore+json. If then the target resource is a data resource (see Section 3.5), then the message body MUST be either application/yang.data+xml or application/yang.data+json.
Plain patch can used to create or update, but not delete, a child resource within the target resource. Please see [I-D.ietf-netconf-yang-patch] for an alternate media-type supporting more granular control. The YANG Patch Media Type allows multiple sub-operations (e.g., merge, delete) within a single PATCH operation.

Example:

To replace just the "year" field in the "album" resource (instead of replacing the entire resource with the PUT method), the client might send a plain patch as follows. Note that the request-line is wrapped for display purposes only:

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

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

If the field is updated, the server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:49:30 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 17:49:30 GMT
ETag: b2788923da4c
```

4.7. DELETE

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

If the user is not authorized to delete the target resource then an error response containing a "403 Forbidden" or "404 Not Found" status-line is returned to the client. All other error responses are handled according to the procedures defined in Section 7.

Example:

To delete a resource such as the "album" resource, the client might send:
DELETE /restconf/data/example-jukebox:jukebox/library/artist=Foo%20Fighters/album=Wasting%20Light HTTP/1.1
Host: example.com

If the resource is deleted, the server might respond:

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

4.8. Query Parameters

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

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

RESTCONF Query Parameters

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

Refer to Appendix D.3 for examples of query parameter usage.
If vendors define additional query parameters, they SHOULD use a prefix (such as the enterprise or organization name) for query parameter names in order to avoid collisions with other parameters.

4.8.1. The "content" Query Parameter

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

The allowed values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>config</td>
<td>Return only configuration descendant data nodes</td>
</tr>
<tr>
<td>nonconfig</td>
<td>Return only non-configuration descendant data nodes</td>
</tr>
<tr>
<td>all</td>
<td>Return all descendant data nodes</td>
</tr>
</tbody>
</table>

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

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

This query parameter MUST be supported by the server.

4.8.2. The "depth" Query Parameter

The "depth" parameter is used to specify the number of nest levels returned in a response for a GET method. The first nest-level consists of the requested data node itself. If the "fields" parameter (Section 4.8.3) is used to select descendant data nodes, these nodes all have a depth value of 1. Any child nodes which are contained within a parent node have a depth value that is 1 greater than its parent.

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

This parameter is only allowed for GET methods on API, datastore, and data resources. A 400 Bad Request error is returned if it used for other methods or resource types.
By default, the server will include all sub-resources within a retrieved resource, which have the same resource type as the requested resource. Only one level of sub-resources with a different media type than the target resource will be returned.

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

4.8.3. The "fields" Query Parameter

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

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

fields-expr = path '(' fields-expr ')' / path ';' fields-expr / path

path = api-identifier [ '/' path ]

"api-identifier" is defined in Section 3.5.1.1.

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

Parentheses are used to specify sub-selectors of a node.

For example, assume the target resource is the 'album' list. To retrieve only the "label" and "catalogue-number" of the "admin" container within an album, use: "fields=admin(label;catalogue-number)".

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

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

If the "fields" query parameter URI is listed in the "capability" leaf-list in Section 9.3, then the server supports the "fields" parameter.
4.8.4. The "insert" Query Parameter

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

The allowed values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
<td>Insert the new data as the new first entry.</td>
</tr>
<tr>
<td>last</td>
<td>Insert the new data as the new last entry.</td>
</tr>
<tr>
<td>before</td>
<td>Insert the new data before the insertion point, as specified by the value of the &quot;point&quot; parameter.</td>
</tr>
<tr>
<td>after</td>
<td>Insert the new data after the insertion point, as specified by the value of the &quot;point&quot; parameter.</td>
</tr>
</tbody>
</table>

The default value is "last".

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

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

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

4.8.5. The "point" Query Parameter

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

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

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

If the "insert" query parameter is not present, or has a value other than "before" or "after", then a 400 Bad Request error is returned.
This parameter contains the instance identifier of the resource to be used as the insertion point for a POST or PUT method.

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

4.8.6. The "filter" Query Parameter

The "filter" parameter is used to indicate which subset of all possible events are of interest. If not present, all events not precluded by other parameters will be sent.

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

The format of this parameter is an XPath 1.0 expression, and is evaluated in the following context:

- 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.
- The set of variable bindings is empty.
- The context node is the root node.

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

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

4.8.7. The "start-time" Query Parameter

The "start-time" parameter is used to trigger the notification replay feature and indicate that the replay should start at the time specified. If the stream does not support replay, per the "replay-support" attribute returned by stream list entry for the stream resource, then the server MUST return the HTTP error code 400 Bad Request.
The value of the "start-time" parameter is of type "date-and-time", defined in the "ietf-yang" YANG module [RFC6991].

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

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

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

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

4.8.8. The "stop-time" Query Parameter

The "stop-time" parameter is used with the replay feature to indicate the newest notifications of interest. This parameter MUST be used with and have a value later than the "start-time" parameter.

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

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

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

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

If the "replay-support" leaf is present in the "stream" entry (defined in Section 9.3) then the server MUST support the "start-time" and "stop-time" query parameters for that stream.
4.8.9. The "with-defaults" Query Parameter

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>report-all</td>
<td>All data nodes are reported</td>
</tr>
<tr>
<td>trim</td>
<td>Data nodes set to the YANG default are not reported</td>
</tr>
<tr>
<td>explicit</td>
<td>Data nodes set by the client are not reported</td>
</tr>
<tr>
<td>report-all-tagged</td>
<td>All data nodes are reported and defaults are tagged</td>
</tr>
</tbody>
</table>

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

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

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

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

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

If the server includes the "with-defaults" query parameter URI in the "capability" leaf-list in Section 9.3, then the "with-defaults" query parameter MUST be supported.
5. Messages

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

5.1. Request URI Structure

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

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

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

```
^       ^        ^       ^         ^
method  entry  resource  query    fragment
M       M        O        O         I
M=mandatory, O=optional, I=ignored
```

<text> replaced by client with real values

- 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 operation. If this field is not present, then the target resource is the API itself, represented by the media type "application/yang.api".

- query: the set of parameters associated with the RESTCONF message. These have the familiar form of "name=value" pairs. Most query parameters are optional to implement by the server and optional to use by the client. Each optional query parameter is identified by a URI. The server MUST list the optional query parameter URIs it supports in the "capabilities" list defined in Section 9.3.
There is a specific set of parameters defined, although the server MAY choose to support query parameters not defined in this document. The contents of the any query parameter value MUST be encoded according to [RFC3986], Section 3.4. Any reserved characters MUST be percent-encoded, according to [RFC3986], section 2.1.

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

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

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

5.2. Message Headers

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

HTTP defines which header lines are required for particular circumstances. Refer to each operation definition section in Section 4 for examples on how particular headers are used.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>Response Content-Types that are acceptable</td>
</tr>
<tr>
<td>Content-Type</td>
<td>The media type of the request body</td>
</tr>
<tr>
<td>Host</td>
<td>The host address of the server</td>
</tr>
<tr>
<td>If-Match</td>
<td>Only perform the action if the entity matches ETag</td>
</tr>
<tr>
<td>If-Modified-Since</td>
<td>Only perform the action if modified since time</td>
</tr>
<tr>
<td>If-Unmodified-Since</td>
<td>Only perform the action if unmodified since time</td>
</tr>
</tbody>
</table>

RESTCONF Request Headers

The following tables summarize the headers most relevant in RESTCONF message responses:
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow</td>
<td>Valid actions when 405 error returned</td>
</tr>
<tr>
<td>Cache-Control</td>
<td>The cache control parameters for the response</td>
</tr>
<tr>
<td>Content-Type</td>
<td>The media type of the response message-body</td>
</tr>
<tr>
<td>Date</td>
<td>The date and time the message was sent</td>
</tr>
<tr>
<td>ETag</td>
<td>An identifier for a specific version of a resource</td>
</tr>
<tr>
<td>Last-Modified</td>
<td>The last modified date and time of a resource</td>
</tr>
<tr>
<td>Location</td>
<td>The resource identifier for a newly created resource</td>
</tr>
</tbody>
</table>

**RESTCONF Response Headers**

5.3. Message Encoding

RESTCONF messages are encoded in HTTP according to [RFC7230]. The "utf-8" character set is used for all messages. RESTCONF message content is sent in the HTTP message-body.

Content is encoded in either JSON or XML format. A server MUST support XML or JSON encoding. XML encoding rules for data nodes are defined in [I-D.ietf-netmod-rfc6020bis]. The same encoding rules are used for all XML content. JSON encoding rules are defined in [I-D.ietf-netmod-yang-json]. JSON encoding of meta-data is defined in [I-D.ietf-netmod-yang-metadata]. This encoding is valid JSON, but also has special encoding rules to identify module namespaces and provide consistent type processing of YANG data.

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

The server MUST support the "Accept" header and "406 Not Acceptable" status code, as defined in [RFC7231]. Response output content encoding format is identified with the Accept header in the request. If is not specified, the request input encoding format is used. 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.

5.4. RESTCONF Meta-Data
The RESTCONF protocol needs to retrieve the same meta-data that is used in the NETCONF protocol. Information about default leafs, last-modified timestamps, etc. are commonly used to annotate representations of the datastore contents. This meta-data is not defined in the YANG schema because it applies to the datastore, and is common across all data nodes.

This information is encoded as attributes in XML. JSON encoding of meta-data is defined in [I-D.ietf-netmod-yang-metadata].

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

### 5.4.1. XML MetaData Encoding Example

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

The server might respond as follows.

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

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

### 5.4.2. JSON MetaData Encoding Example

Note that RFC 6243 defines the "default" attribute with XSD, not YANG, so the YANG module name has to be assigned manually. The value "ietf-netconf-with-defaults" is assigned for JSON meta-data encoding.

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

The server might respond as follows.

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

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

5.5. Return Status

Each message represents some sort of resource access. An HTTP "status-line" header line is returned for each request. If a 4xx or 5xx range status code is returned in the status-line, then the error information will be returned in the response, according to the format defined in Section 7.1.

5.6. Message Caching

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

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

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


6.1. Server Support

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

6.2. Event Streams

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

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

For example:

The client might send the following request:

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

The server might send the following response:

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

<streams
  xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring">
  <stream>
    <name>NETCONF</name>
    <description>default NETCONF event stream</description>
    <replay-support>true</replay-support>
  </stream>
</streams>
<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>
</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>
    2007-01T00:00:00Z
  </replay-log-creation-time>
  <access>
    <encoding>xml</encoding>
    <location>https://example.com/streams/syslog-critical</location>
  </access>
</stream>
</streams>

6.3. Subscribing to Receive Notifications

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

The server will treat the connection as an event stream, using the Server Sent Events [W3C.CR-eventsource-20121211] transport strategy.

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

For example:

The client might send the following request:

GET /restconf/data/ietf-restconf-monitoring:restconf-state/
  streams/stream=NETCONF/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.api+xml

<location
  xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring">
  https://example.com/streams/NETCONF
</location>

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

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

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

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

The server SHOULD support the "NETCONF" notification stream defined in [RFC5277]. For this stream, RESTCONF notification subscription requests MAY specify parameters indicating the events it wishes to receive. These query parameters are optional to implement, and only available if the server supports them.

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

**NETCONF Stream Query Parameters**

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

Refer to Appendix D.3.6 for filter parameter examples.

6.4. Receiving Event Notifications

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

The structure of the event data is based on the "notification" element definition in Section 4 of [RFC5277]. It MUST conform to the schema for the "notification" element in Section 4 of [RFC5277], except the XML namespace for this element is defined as:

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

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

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

In the following example, the YANG module "example-mod" is used:
module example-mod {
    namespace "http://example.com/event/1.0";

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

An example SSE event notification encoded using XML:

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

An example SSE event notification encoded using JSON:

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

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

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

XML:

data: <notification xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf" event-time="2013-12-21T00:01:00Z"></notification>

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

7. Error Reporting

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

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

<table>
<thead>
<tr>
<th>Status-Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 Continue</td>
<td>POST accepted, 201 should follow</td>
</tr>
<tr>
<td>200 OK</td>
<td>Success with response message-body</td>
</tr>
<tr>
<td>201 Created</td>
<td>POST to create a resource success</td>
</tr>
<tr>
<td>202 Accepted</td>
<td>POST to create a resource accepted</td>
</tr>
<tr>
<td>204 No Content</td>
<td>Success without response message-body</td>
</tr>
<tr>
<td>304 Not Modified</td>
<td>Conditional operation not done</td>
</tr>
<tr>
<td>400 Bad Request</td>
<td>Invalid request message</td>
</tr>
<tr>
<td>403 Forbidden</td>
<td>Access to resource denied</td>
</tr>
<tr>
<td>404 Not Found</td>
<td>Resource target or resource node not found</td>
</tr>
<tr>
<td>405 Method Not Allowed</td>
<td>Method not allowed for target</td>
</tr>
</tbody>
</table>
HTTP Status Codes used in RESTCONF

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

<table>
<thead>
<tr>
<th>&lt;error-tag&gt;</th>
<th>status code</th>
</tr>
</thead>
<tbody>
<tr>
<td>in-use</td>
<td>409</td>
</tr>
<tr>
<td>invalid-value</td>
<td>400</td>
</tr>
<tr>
<td>too-big</td>
<td>413</td>
</tr>
<tr>
<td>missing-attribute</td>
<td>400</td>
</tr>
<tr>
<td>bad-attribute</td>
<td>400</td>
</tr>
<tr>
<td>unknown-attribute</td>
<td>400</td>
</tr>
<tr>
<td>bad-element</td>
<td>400</td>
</tr>
<tr>
<td>unknown-element</td>
<td>400</td>
</tr>
<tr>
<td>unknown-namespace</td>
<td>400</td>
</tr>
<tr>
<td>access-denied</td>
<td>403</td>
</tr>
<tr>
<td>lock-denied</td>
<td>409</td>
</tr>
<tr>
<td>resource-denied</td>
<td>409</td>
</tr>
<tr>
<td>rollback-failed</td>
<td>500</td>
</tr>
<tr>
<td>data-exists</td>
<td>409</td>
</tr>
<tr>
<td>data-missing</td>
<td>409</td>
</tr>
<tr>
<td>operation-not-supported</td>
<td>501</td>
</tr>
<tr>
<td>operation-failed</td>
<td>500</td>
</tr>
<tr>
<td>partial-operation</td>
<td>500</td>
</tr>
<tr>
<td>malformed-message</td>
<td>400</td>
</tr>
</tbody>
</table>

Mapping from error-tag to status code
7.1. Error Response Message

When an error occurs for a request message on a data resource or an operation resource, and a "4xx" class of status codes (except for status code "403 Forbidden"), then the server SHOULD send a response message-body containing the information described by the "errors" container definition within the YANG module Section 8. The Content-Type of this response message MUST be application/yang.errors (see example below).

The client MAY specify the desired encoding for error messages by specifying the appropriate media-type in the Accept header. If no error media is specified, then the media type of the request message is used. If there is no request message the server MUST select "application/yang.errors+xml" or "application/yang.errors+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:

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

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

Examples:

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

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

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

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

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

The client might send:

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

The server might respond (some lines wrapped for display purposes):

HTTP/1.1 409 Conflict
Date: Mon, 23 Apr 2012 17:11:00 GMT
Server: example-server
Content-Type: application/yang.errors+xml
<errors xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <error>
    <error-type>protocol</error-type>
    <error-tag>data-exists</error-tag>
    <error-path>
      xmlns:jb="https://example.com/ns/example-jukebox"
      /rc:restconf/rc:data/jb:jukebox
    </error-path>
    <error-message>
      Data already exists, cannot create new resource
    </error-message>
  </error>
</errors>

8. RESTCONF module

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

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

<CODE BEGINS> file "ietf-restconf@2015-10-18.yang"

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

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

  contact
    "WG Web:  <http://tools.ietf.org/wg/netconf/>
    WG List:  <mailto:netconf@ietf.org>
    WG Chair: Mehmet Ersue
      <mailto:mehmet.ersue@nsn.com>
    WG Chair: Mahesh Jethanandani
      <mailto:mjethanandani@gmail.com>
    Editor:  Andy Bierman
      <mailto:andy@yumaworks.com>
This module contains conceptual YANG specifications for basic RESTCONF media type definitions used in RESTCONF protocol messages.

Note that the YANG definitions within this module do not represent configuration data of any kind. The 'restconf-media-type' YANG extension statement provides a normative syntax for XML and JSON message encoding purposes.

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

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

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

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

// RFC Ed.: remove this note
// Note: extracted from draft-ietf-netconf-restconf-08.txt

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
revision 2015-10-18 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: RESTCONF Protocol.";
}

extension restconf-media-type {
  argument media-type-id {
    yin-element true;
This extension is used to specify a YANG data structure which represents a conceptual RESTCONF media type. Data definition statements within this extension specify the generic syntax for the specific media type.

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

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

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

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

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

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

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

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

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

";
}

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

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

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

carrier 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 layer";
    }
  }
}
description "The rpc or notification layer";
}
enum protocol {
    description "The protocol operation layer";
}
enum application {
    description "The server application layer";
}
}
mandatory true;
description "The protocol layer where the error occurred.";
}
leaf error-tag {
    type string;
    mandatory true;
description "The enumerated error tag.";
}
leaf error-app-tag {
    type string;
description "The application-specific error tag.";
}
leaf error-path {
    type instance-identifier;
description "The YANG instance identifier associated with the error node.";
}
leaf error-message {
    type string;
description "A message describing the error.";
}
anyxml error-info {
    description "This anyxml value MUST represent a container with zero or more data nodes representing additional error information.";
}
}
grouping restconf {
    description
    "Conceptual container representing the application/yang.api resource type."
}

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

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

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

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

    E.g.;

    POST /restconf/operations/show-log-errors

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

}
9. RESTCONF Monitoring

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

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

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

9.1. restconf-state/capabilities

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

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

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

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

The server MUST include a "capability" URI leaf-list entry identifying each supported optional protocol feature. This includes optional query parameters and MAY include other capability URIs defined outside this document.
9.1.1. Query Parameter URIs

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Section</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>depth</td>
<td>4.8.2</td>
<td>urn:ietf:params:restconf:capability:depth:1.0</td>
</tr>
<tr>
<td>fields</td>
<td>4.8.3</td>
<td>urn:ietf:params:restconf:capability:fields:1.0</td>
</tr>
<tr>
<td>filter</td>
<td>4.8.6</td>
<td>urn:ietf:params:restconf:capability:filter:1.0</td>
</tr>
<tr>
<td>replay</td>
<td>4.8.7</td>
<td>urn:ietf:params:restconf:capability:replay:1.0</td>
</tr>
<tr>
<td>with-</td>
<td>4.8.8</td>
<td>defaults:1.0</td>
</tr>
<tr>
<td>defaults</td>
<td>4.8.9</td>
<td></td>
</tr>
</tbody>
</table>

RESTCONF Query Parameter URIs

9.1.2. The "defaults" Protocol Capability URI

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

<table>
<thead>
<tr>
<th>Name</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaults</td>
<td>urn:ietf:params:restconf:capability:defaults:1.0</td>
</tr>
</tbody>
</table>

RESTCONF defaults capability URI

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

| Value     | Description |
### report-all
No data nodes are considered default

### trim
Values set to the YANG default-stmt value are default

### explicit
Values set by the client are never considered default

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

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

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

Example: (split for display purposes only)

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

#### 9.2. restconf-state/streams

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

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

Stream resources are defined in Section 3.8. Notifications are defined in Section 6.

#### 9.3. RESTCONF Monitoring Module

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

The "ietf-yang-types" and "ietf-inet-types" modules from [RFC6991] are used by this module for some type definitions.

RFC Ed.: update the date below with the date of RFC publication and remove this note.
<CODE BEGINS> file "ietf-restconf-monitoring@2015-06-19.yang"

module ietf-restconf-monitoring {
    prefix "rcmon";

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

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

    contact
        "WG Web:  <http://tools.ietf.org/wg/netconf/>
        WG List:  <mailto:netconf@ietf.org>
        WG Chair: Mehmet Ersue
                    <mailto:mehmet.ersue@nsn.com>
        WG Chair: Maheesh Jethanandani
                    <mailto:mjethanandani@gmail.com>
        Editor: Andy Bierman
                    <mailto:andy@yumaworks.com>
        Editor: Martin Bjorklund
                    <mailto:mbj@tail-f.com>
        Editor: Kent Watsen
                    <mailto:kwatsen@juniper.net>"

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

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

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

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

revision 2015-06-19 {
    description "Initial revision.";
    reference "RFC XXXX: RESTCONF Protocol.";
}

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

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

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

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

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

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

leaf replay-support {
  type boolean;
  description
    "Indicates if replay buffer supported for this stream. If 'true',
     then the server MUST support the 'start-time' and 'stop-time'
     query parameters for this stream.";
  reference
    "RFC 5277, Section 3.4, <replaySupport> element.";
}

leaf replay-log-creation-time {
  when "../replay-support" {
    description
      "Only present if notification replay is supported";
  }
  type yang:date-and-time;
  description
    "Indicates the time the replay log for this stream was created.";
  reference
    "RFC 5277, Section 3.4, <replayLogCreationTime> element.";
}

list access {
  key 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
     'application/yang.stream' is expected for all event streams. This
     list identifies the sub-types supported for this stream."
  }

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

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

<CODE ENDS>

10. YANG Module Library

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

10.1. modules

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

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

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

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

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

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

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

11.  IANA Considerations

11.1.  The "restconf" Relation Type

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

Relation Name:  restconf

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

Reference:  RFC XXXX

11.2.  YANG Module Registry

This document registers two URIs in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made.

Registrant Contact: The NETMOD WG of the IETF.
XML: N/A, the requested URI is an XML namespace.
This document registers two YANG modules in the YANG Module Names registry [RFC6020].

name:             ietf-restconf
prefix:           rc
reference:        RFC XXXX

name:             ietf-restconf-monitoring
prefix:           rcmon
reference:        RFC XXXX

11.3.  application/yang Media Sub Types

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

- api
- data
- datastore
- errors
- operation
- stream

Type name: application

Subtype name: yang.xxx

Required parameters: none

Optional parameters: See section 4.8 in RFC XXXX

Encoding considerations: 8-bit

Security considerations: See Section 12 in RFC XXXX

Interoperability considerations: none

// RFC Ed.: replace XXXX with RFC number and remove this note
Published specification: RFC XXXX
11.4. RESTCONF Capability URNs

[Note to RFC Editor:  
The RESTCONF Protocol Capability Registry does not yet exist;  
Need to ask IANA to create it; remove this note for publication]

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

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

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

<table>
<thead>
<tr>
<th>Index</th>
<th>Capability Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>:defaults</td>
<td>urn:ietf:params:restconf:capability:defaults:1.0</td>
</tr>
<tr>
<td>:depth</td>
<td>urn:ietf:params:restconf:capability:depth:1.0</td>
</tr>
<tr>
<td>:fields</td>
<td>urn:ietf:params:restconf:capability:fields:1.0</td>
</tr>
<tr>
<td>:filter</td>
<td>urn:ietf:params:restconf:capability:filter:1.0</td>
</tr>
<tr>
<td>:replay</td>
<td>urn:ietf:params:restconf:capability:replay:1.0</td>
</tr>
<tr>
<td>:with-defaults</td>
<td>urn:ietf:params:restconf:capability:with-defaults:1.0</td>
</tr>
</tbody>
</table>

12. Security Considerations

This section provides security considerations for the resources  
defined by the RESTCONF protocol.  Security considerations for HTTPS  
are defined in [RFC2818].  Security considerations for the content  
manipulated by RESTCONF can be found in the documents defining data  
models.
This document does not specify an authentication scheme, but it does require that an authenticated NETCONF username be associated with each HTTP request. The authentication scheme MAY be implemented in the underlying transport layer (e.g., client certificates) or within the HTTP layer (e.g., Basic Auth, OAuth, etc.). RESTCONF does not itself define an authentication mechanism. Authentication MUST occur in a lower layer. Implementors SHOULD provide a comprehensive authorization scheme with RESTCONF and ensure that the resulting NETCONF username is made available to the RESTCONF server.

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

Configuration information is by its very nature sensitive. Its transmission in the clear and without integrity checking leaves devices open to classic eavesdropping and false data injection attacks. Configuration information often contains passwords, user names, service descriptions, and topological information, all of which are sensitive. Because of this, this protocol SHOULD be implemented carefully with adequate attention to all manner of attack one might expect to experience with other management interfaces.

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

13. Acknowledgements

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

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

14.1. Normative References

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

[I-D.ietf-netmod-rfc6020bis]

[I-D.ietf-netmod-yang-json]
Lhotka, L., "JSON Encoding of Data Modeled with YANG", draft-ietf-netmod-yang-json-06 (work in progress), October 2015.

[I-D.ietf-netmod-yang-metadata]
Lhotka, L., "Defining and Using Metadata with YANG", draft-ietf-netmod-yang-metadata-02 (work in progress), September 2015.


[draft-thomson-httpbis-cant-01]
14.2. Informative References

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

[rest-dissertation]

Appendix A. Change Log

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

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

A.1. 07 - 08
  o add support for YANG 1.1 action statement
  o changed mandatory encoding from XML to XML or JSON
  o fix syntax in fields parameter definition
  o add meta-data encoding examples for XML and JSON
  o remove RFC 2396 references and update with 3986
  o change encoding of a key so quoted string are not used, since they are already percent-encoded. A zero-length string is not encoded (/list=foo,,baz)
  o Add example of percent-encoded key value

A.2. 06 - 07
  o fixed all issues identified in email from Jernej Tuljak in netconf email 2015-06-22
  o fixed error example bug where error-urlpath was still used. Changed to error-path.
- added mention of YANG Patch and informative reference
- added support for YANG 1.1, specifically support for anydata and actions
- removed the special field value "*", since it is no longer needed

A.3. 05 - 06
- fixed RESTCONF issue #23 (ietf-restconf-monitoring bug)

A.4. 04 - 05
- changed term 'notification event' to 'event notification'
- removed intro text about framework and meta-model
- removed early mention of API resources
- removed term unified datastore and cleaned up text about NETCONF datastores
- removed text about not immediate persistence of edits
- removed RESTCONF-specific data-resource-identifier typedef and its usage
- clarified encoding of key leaves
- changed several examples from JSON to XML encoding
- made 'insert' and 'point' query parameters mandatory to implement
- removed ":insert" capability URI
- renamed stream/encoding to stream/access
- renamed stream/encoding/type to stream/access/encoding
- renamed stream/encoding/events to stream/access/location
- changed XPath from informative to normative reference
- changed rest-dissertation from normative to informative reference
- changed example-jukebox playlist 'id' from a data-resource-identifier to a leafref pointing at the song name
A.5. 03 – 04

- renamed 'select' to 'fields' (#1)
- moved collection resource and page capability to draft-ietf-netconf-restconf-collection-00 (#3)
- added mandatory "defaults" protocol capability URI (#4)
- added optional "with-defaults" query parameter URI (#4)
- clarified authentication procedure (#9)
- moved ietf-yang-library module to draft-ietf-netconf-yang-library-00 (#13)
- clarified that JSON encoding of module name in a URI MUST follow the netmod-yang-json encoding rules (#14)
- added restconf-media-type extension (#15)
- remove "content" query parameter URI and made this parameter mandatory (#16)
- clarified datastore usage
- changed lock-denied error example
- added with-defaults query parameter example
- added term "RESTCONF Capability"
- changed NETCONF Capability URI registry usage to new RESTCONF Capability URI Registry usage

A.6. 02 – 03

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

A.7. 01 - 02

- moved query parameter definitions from the YANG module back to the plain text sections
- made all query parameters optional to implement
- defined query parameter capability URI
- moved 'streams' to new YANG module (ietf-restconf-monitoring)
- added 'capabilities' container to new YANG module (ietf-restconf-monitoring)
- moved 'modules' container to new YANG module (ietf-yang-library)
- added new leaf 'module-set-id' (ietf-yang-library)
- added new leaf 'conformance' (ietf-yang-library)
- changed 'schema' leaf to type inet:uri that returns the location of the YANG schema (instead of returning the schema directly)
- changed 'events' leaf to type inet:uri that returns the location of the event stream resource (instead of returning events directly)
- changed examples for yang.api resource since the monitoring information is no longer in this resource
- closed issue #1 'select parameter' since no objections to the proposed syntax
- closed "encoding of list keys" issue since no objection to new encoding of list keys in a target resource URI.
- moved open issues list to the issue tracker on github
A.8.  00 - 01

- fixed content=nonconfig example (non-config was incorrect)
- closed open issue ‘message-id’. There is no need for a message-id field, and RFC 2392 does not apply.
- closed open issue ‘server support verification’. The headers used by RESTCONF are widely supported.
- removed encoding rules from section on RESTCONF Meta-Data. This is now defined in "I-D.lhotka-netmod-yang-json".
- added media type application/yang.errors to map to errors YANG grouping. Updated error examples to use new media type.
- closed open issue ‘additional datastores’. Support may be added in the future to identify new datastores.
- closed open issue ‘PATCH media type discovery’. The section on PATCH has an added sentence on the Accept-Patch header.
- closed open issue ‘YANG to resource mapping’. Current mapping of all data nodes to resources will be used in order to allow mandatory DELETE support. The PATCH operation is optional, as well as the YANG Patch media type.
- closed open issue ‘_self links for HATEOAS support’. It was decided that they are redundant because they can be derived from the YANG module for the specific data.
- added explanatory text for the ‘select’ parameter.
- added RESTCONF Path Resolution section for discovering the root of the RESTCONF API using the /.well-known/host-meta.
- added an "error" media type to for structured error messages
- added Secure Transport section requiring TLS
- added Security Considerations section
- removed all references to "REST-like"

A.9.  bierman:restconf-04 to ietf:restconf-00

- updated open issues section
Appendix B.  Open Issues

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

The RESTCONF issues are tracked on github.com:
https://github.com/netconf-wg/restconf/issues

Appendix C.  Example YANG Module

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

YANG Tree Diagram for "example-jukebox" Module

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

rpcs:
C.1. example-jukebox YANG Module

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

    organization "Example, Inc.";
    contact "support at example.com";
    description "Example Jukebox Data Model Module";
    revision "2015-04-04" {
        description "Initial version.";
        reference "example.com document 1-4673";
    }

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

    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.";

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.";
}

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

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

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

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

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

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

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

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

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

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

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

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

list playlist {
    key name;

description "Example configuration data resource";

leaf name {
    type string;
    description "The name of the playlist.";
}

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

description
  "Example nested configuration data resource";

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

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

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

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

Appendix D. RESTCONF Message Examples

The examples within this document use the normative YANG module defined in Section 8 and the non-normative example YANG module defined in Appendix C.1.

This section shows some typical RESTCONF message exchanges.

D.1. Resource Retrieval Examples

D.1.1. Retrieve the Top-level API Resource

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

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

The server might respond as follows:

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

{
  "ietf-restconf:restconf": {
    "data" : [ null ],
    "operations" : [ null ]
  }
}

To request that the response content to be encoded in XML, the "Accept" header can be used, as in this example request:
GET /restconf HTTP/1.1
Host: example.com
Accept: application/yang.api+xml

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

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

<restconf xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <data/>
  <operations/>
</restconf>

D.1.2. Retrieve The Server Module Information

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

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

The server might respond as follows.

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

D.1.3. Retrieve The Server Capability Information

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

GET /restconf/data/ietf-restconf-monitoring:restconf-state/capabilities HTTP/1.1
Host: example.com
Accept: application/yang.data+xml
The server might respond as follows. The extra whitespace in ‘capability’ elements for display purposes only.

HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:02:00 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Last-Modified: Sun, 22 Apr 2012 01:00:14 GMT
Content-Type: application/yang.data+xml

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

D.2.  Edit Resource Examples

D.2.1.  Create New Data Resources

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

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

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

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

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

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

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

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

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

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

D.2.2. Detect Resource Entity Tag Change
In this example, the server just supports the mandatory datastore last-changed timestamp. The client has previously retrieved the "Last-Modified" header and has some value cached to provide in the following request to patch an "album" list entry with key value "Wasting Light". Only the "year" field is being updated.

```
PATCH /restconf/data/example-jukebox:jukebox/
    library/artist=Foo%20Fighters/album=Wasting%20Light/year
HTTP/1.1
Host: example.com
If-Unmodified-Since: Mon, 23 Apr 2012 17:01:00 GMT
Content-Type: application/yang.data+json
{
    "example-jukebox:year": "2011"
}
```

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

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

D.2.3. Edit a Datastore Resource

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

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

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

D.3. Query Parameter Examples

D.3.1. "content" Parameter

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

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

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

Example 1: content=all

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

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

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json
```
Example 2: content=config

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

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

The server might respond:

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

{  
  "example-events:events" : { 
  "event" : [ 
    { 
      "name" : "interface-up", 
      "description" : "Interface up notification count", 
      "event-count" : 42 
    }, 
    { 
      "name" : "interface-down", 
      "description" : "Interface down notification count", 
      "event-count" : 4 
    } 
  ] 
  } 
}
Example 3: content=nonconfig

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

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

The server might respond:

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

{
  "example-events:events" : {
    "event" : [
      {
        "name" : "interface-up",
        "event-count" : 42
      },
      {
        "name" : "interface-down",
        "event-count" : 4
      }
    ]
  }
}
D.3.2. "depth" Parameter

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

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

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

Example 1: depth=unbounded

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

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

The server might respond:

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

{
  "example-jukebox:jukebox" : {
    "library" : {
      "artist" : [
        {
          "name" : "Foo Fighters",
          "album" : {
            "name" : "Wasting Light",
            "genre" : "example-jukebox:alternative",
            "year" : 2011,
            "song" : [
              {
                "name" : "Wasting Light",
              }
            ]
          }
        }
      ]
    }
  }
}
"location" : 
    "/media/foo/a7/wasting-light.mp3",
"format" : "MP3",
"length" : 286
},

{ "name" : "Rope",
"location" : "/media/foo/a7/rope.mp3",
"format" : "MP3",
"length" : 259
}
]
}
]
"playlist" : [ 
{ "name" : "Foo-One",
"description" : "example playlist 1",
"song" : [ 
{ "index" : 1,
"id" : "https://example.com/restconf/data/example-jukebox:jukebox/library/artist=
Foo%20Fighters/album=Wasting%20Light/song=Rope"
},
{ "index" : 2,
"id" : "https://example.com/restconf/data/example-jukebox:jukebox/library/artist=
Foo%20Fighters/album=Wasting%20Light/song=Bridge%20Burning"
}
]
}
,"player" : { "gap" : 0.5
}
]

Example 2: depth=1
To determine if 1 or more resource instances exist for a given target resource, the value "1" is used.

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

The server might respond:

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

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

Example 3: depth=3

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

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

The server might respond:

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

{
   "example-jukebox:jukebox" : {
      "library" : {
         "artist" : [ null ]
      },
      "playlist" : [
         {
            "name" : "Foo-One",
            "description" : "example playlist 1",
            "song" : [ null ]
         }
      ]
   }
}
D.3.3. "fields" Parameter

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

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

The server might respond as follows.

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

{
  "ietf-yang-library:modules": {
    "module": [
      {
        "name" : "example-jukebox",
        "revision" : "2015-06-04"
      },
      {
        "name" : "ietf-inet-types",
        "revision" : "2013-07-15"
      },
      {
        "name" : "ietf-restconf-monitoring",
        "revision" : "2015-06-19"
      },
      {
        "name" : "ietf-yang-library",
        "revision" : "2015-07-03"
      },
      {
        "name" : "ietf-yang-types",
        "revision" : "2013-07-15"
      }
    ]
  }
}
D.3.4. "insert" Parameter

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

Request from client:

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

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

Response from server:

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

D.3.5. "point" Parameter

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

Request from client:

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

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

Response from server:
HTTP/1.1 204 No Content


D.3.6. "filter" Parameter

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

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

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

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

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

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

// Note: the module name is used as prefix.
// filter = (/example-mod:event1/name='joe' and
//           /example-mod:event1/status='online')
GET /mystreams/NETCONF?
    filter=(%2Fexample-mod%3Aevent1%2Fname%3D'joe'%20and
           %20%2Fexample-mod%3Aevent1%2Fstatus%3D'online')
D.3.7. "start-time" Parameter

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

D.3.8. "stop-time" Parameter

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

D.3.9. "with-defaults" Parameter

The following YANG module is assumed for this example.

```yang
module example-interface {
  prefix "exif";
  namespace "urn:example.com:params:xml:ns:yang:example-interface";

  container interfaces {
    list interface {
      key name;
      leaf name { type string; }
      leaf mtu { type uint32; }
      leaf status {
        config false;
        type enumeration {
          enum up;
          enum down;
          enum testing;
        }
      }
    }
  }
}
```

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

Without query parameter:

GET /restconf/data/example:interfaces/interface=eth1 HTTP/1.1
Host: example.com
Accept: application/yang.data+json

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

{
    "example:interface": [
        {
            "name" : "eth1",
            "status" : "up"
        }
    ]
}

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

With query parameter:

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

The server might respond as follows.

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

{
    "example:interface": [
        {
            "name" : "eth1",
            "mtu" : 1500,
            "status" : "up"
        }
    ]
}

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

Authors’ Addresses
Andy Bierman
YumaWorks
Email: andy@yumaworks.com

Martin Bjorklund
Tail-f Systems
Email: mbj@tail-f.com

Kent Watsen
Juniper Networks
Email: kwatsen@juniper.net
NETCONF Server and RESTCONF Server Configuration Models
draft-ietf-netconf-server-model-08

Abstract

This draft defines a NETCONF server configuration data model and a
RESTCONF server configuration data model. These data models enable
configuration of the NETCONF and RESTCONF services themselves,
including which transports are supported, what ports the servers
listen on, call-home parameters, client authentication, and related
parameters.

Editorial Note (To be removed by RFC Editor)

This draft contains many placeholder values that need to be replaced
with finalized values at the time of publication. This note
summarizes all of the substitutions that are needed. Please note
that no other RFC Editor instructions are specified anywhere else in
this document.

This document contains references to other drafts in progress, both
in the Normative References section, as well as in body text
throughout. Please update the following references to reflect their
final RFC assignments:

- draft-ietf-netconf-restconf
- draft-ietf-netconf-call-home

Artwork in this document contains shorthand references to drafts in
progress. Please apply the following replacements:

- "VVVV" -- the assigned RFC value for this draft
- "XXXX" -- the assigned RFC value for draft-ietf-netconf-restconf
- "YYYY" -- the assigned RFC value for draft-ietf-netconf-call-home

Artwork in this document contains placeholder values for ports
pending IANA assignment from "draft-ietf-netconf-call-home". Please
apply the following replacements:
o "7777" --> the assigned port value for "netconf-ch-ssh"

o "8888" --> the assigned port value for "netconf-ch-tls"

o "9999" --> the assigned port value for "restconf-ch-tls"

Artwork in this document contains placeholder values for the date of publication of this draft. Please apply the following replacement:

o "2015-10-09" --> the publication date of this draft

The following two Appendix sections are to be removed prior to publication:

o Appendix B. Change Log

o Appendix C. Open Issues

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 11, 2016.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction ............................................. 4
   1.1. Terminology ........................................ 4
   1.2. Tree Diagrams ........................................ 4
2. Objectives ................................................. 5
   2.1. Support all NETCONF and RESTCONF transports ....... 5
   2.2. Enable each transport to select which keys to use .... 5
   2.3. Support authenticating NETCONF/RESTCONF clients
        certificates .......................................... 5
   2.4. Support mapping authenticated NETCONF/RESTCONF client
        certificates to usernames ............................... 5
   2.5. Support both listening for connections and call home ... 6
   2.6. For Call Home connections ............................. 6
       2.6.1. Support more than one NETCONF/RESTCONF client ... 6
       2.6.2. Support NETCONF/RESTCONF clients having more than one
                endpoint ......................................... 6
       2.6.3. Support a reconnection strategy .................. 6
       2.6.4. Support both persistent and periodic connections .. 6
       2.6.5. Reconnection strategy for periodic connections ... 7
       2.6.6. Keep-alives for persistent connections .......... 7
       2.6.7. Customizations for periodic connections .......... 7
3. High-Level Design .......................................... 7
4. Solution ................................................... 8
   4.1. The Keychain Model ................................... 8
       4.1.1. Tree Diagram ................................... 9
       4.1.2. Example Usage ................................ 9
       4.1.3. YANG Model ....................................... 15
   4.2. The SSH Server Model .................................. 20
       4.2.1. Tree Diagram ................................... 21
       4.2.2. Example Usage ................................ 21
       4.2.3. YANG Model ....................................... 22
   4.3. The TLS Server Model .................................. 26
       4.3.1. Tree Diagram ................................... 26
       4.3.2. Example Usage ................................ 27
       4.3.3. YANG Model ....................................... 27
   4.4. The NETCONF Server Model ................................ 31
       4.4.1. Tree Diagram ................................... 31
       4.4.2. Example Usage ................................ 33
       4.4.3. YANG Model ....................................... 37
   4.5. The RESTCONF Server Model ................................ 47
       4.5.1. Tree Diagram ................................... 47
       4.5.2. Example Usage ................................ 49
       4.5.3. YANG Model ....................................... 51
5. Security Considerations ..................................... 59
1. Introduction

This draft defines a NETCONF [RFC6241] server configuration data model and a RESTCONF [draft-ietf-netconf-restconf] server configuration data model. These data models enable configuration of the NETCONF and RESTCONF services themselves, including which transports are supported, what ports the servers listen on, call-home parameters, client authentication, and related parameters.

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Tree Diagrams

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

- Brackets "[" and "]" enclose list keys.
- Braces "{" and "}" enclose feature names, and indicate that the named feature must be present for the subtree to be present.
- Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

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

2. Objectives

The primary purpose of the YANG modules defined herein is to enable the configuration of the NETCONF and RESTCONF services on a network element. This scope includes the following objectives:

2.1. Support all NETCONF and RESTCONF transports

The YANG module should support all current NETCONF and RESTCONF transports, namely NETCONF over SSH [RFC6242], NETCONF over TLS [RFC7589], and RESTCONF over TLS [draft-ietf-netconf-restconf], and to be extensible to support future transports as necessary.

Because implementations may not support all transports, the module should use YANG "feature" statements so that implementations can accurately advertise which transports are supported.

2.2. Enable each transport to select which keys to use

Servers may have a multiplicity of host-keys or server-certificates from which subsets may be selected for specific uses. For instance, a NETCONF server may want to use one set of SSH host-keys when listening on port 830, and a different set of SSH host-keys when calling home. The data models provided herein should enable configuration of which keys to use on a per-use basis.

2.3. Support authenticating NETCONF/RESTCONF clients certificates

When a certificate is used to authenticate a NETCONF or RESTCONF client, there is a need to configure the server to know how to authenticate the certificates. The server should be able to authenticate the client's certificate either by using path-validation to a configured trust anchor or by matching the client-certificate to one previously configured.

2.4. Support mapping authenticated NETCONF/RESTCONF client certificates to usernames

When a client certificate is used for TLS client authentication, the NETCONF/RESTCONF server must be able to derive a username from the authenticated certificate. Thus the modules defined herein should enable this mapping to be configured.
2.5. Support both listening for connections and call home

The NETCONF and RESTCONF protocols were originally defined as having the server opening a port to listen for client connections. More recently the NETCONF working group defined support for call-home ([draft-ietf-netconf-call-home]), enabling the server to initiate the connection to the client, for both the NETCONF and RESTCONF protocols. Thus the modules defined herein should enable configuration for both listening for connections and calling home. Because implementations may not support both listening for connections and calling home, YANG "feature" statements should be used so that implementation can accurately advertise the connection types it supports.

2.6. For Call Home connections

The following objectives only pertain to call home connections.

2.6.1. Support more than one NETCONF/RESTCONF client

A NETCONF/RESTCONF server may be managed by more than one NETCONF/RESTCONF client. For instance, a deployment may have one client for provisioning and another for fault monitoring. Therefore, when it is desired for a server to initiate call home connections, it should be able to do so to more than one client.

2.6.2. Support NETCONF/RESTCONF clients having more than one endpoint

An NETCONF/RESTCONF client managing a NETCONF/RESTCONF server may implement a high-availability strategy employing a multiplicity of active and/or passive endpoint. Therefore, when it is desired for a server to initiate call home connections, it should be able to connect to any of the client’s endpoints.

2.6.3. Support a reconnection strategy

Assuming a NETCONF/RESTCONF client has more than one endpoint, then it becomes necessary to configure how a NETCONF/RESTCONF server should reconnect to the client should it lose its connection to one the client’s endpoints. For instance, the NETCONF/RESTCONF server may start with first endpoint defined in a user-ordered list of endpoints or with the last endpoints it was connected to.

2.6.4. Support both persistent and periodic connections

NETCONF/RESTCONF clients may vary greatly on how frequently they need to interact with a NETCONF/RESTCONF server, how responsive interactions need to be, and how many simultaneous connections they
can support. Some clients may need a persistent connection to servers to optimize real-time interactions, while others prefer periodic interactions in order to minimize resource requirements. Therefore, when it is necessary for server to initiate connections, it should be configurable if the connection is persistent or periodic.

2.6.5. Reconnection strategy for periodic connections

The reconnection strategy should apply to both persistent and periodic connections. How it applies to periodic connections becomes clear when considering that a periodic "connection" is a logical connection to a single server. That is, the periods of unconnectedness are intentional as opposed to due to external reasons. A periodic "connection" should always reconnect to the same server until it is no longer able to, at which time the reconnection strategy guides how to connect to another server.

2.6.6. Keep-alives for persistent connections

If a persistent connection is desired, it is the responsibility of the connection initiator to actively test the "aliveness" of the connection. The connection initiator must immediately work to reestablish a persistent connection as soon as the connection is lost. How often the connection should be tested is driven by NETCONF/RESTCONF client requirements, and therefore keep-alive settings should be configurable on a per-client basis.

2.6.7. Customizations for periodic connections

If a periodic connection is desired, it is necessary for the NETCONF/RESTCONF server to know how often it should connect. This frequency determines the maximum amount of time a NETCONF/RESTCONF client may have to wait to send data to a server. A server may connect to a client before this interval expires if desired (e.g., to send data to a client).

3. High-Level Design

The solution presented in this document defines a configurable keychain object, reusable groupings for SSH and TLS based servers, and, finally, the configurable NETCONF and RESTCONF server objects, which are the primary purpose for this draft. Each of these are defined in a distinct YANG module, thus a total of five YANG modules are defined in this document. The relationship between these five YANG modules is illustrated by the tree diagram below.
4.  Solution

Each of the following five sections relate to one of the YANG modules depicted by the figure above.

4.1.  The Keychain Model

The keychain model depicted in this section provides a configurable object having the following characteristics:

- A semi-configurable list of private keys, each with one or more associated certificates. Though private keys can only be created via an RPC (see bullet #3 below), the entries of the list may be renamed and have certificates associated with them after creation.

- A configurable list of lists of trust anchor certificates. This enables the server to have use-specific trust anchors. For instance, one list of trust anchors might be used to authenticate management connections (e.g., client certificate-based authentication for NETCONF or RESTCONF connections), and a different list of trust anchors might be used for when connecting to a specific Internet-based service (e.g., a zero touch bootstrap server).

- An RPC to request the server to generate a new private key using the specified algorithm and key length.
An RPC to generate a certificate signing request for an existing private key, a passed subject, and an optional attributes. The signed certificate returned from an external certificate authority (CA) can be set using a standard configuration change request (e.g., <edit-config>).

4.1.1. Tree Diagram

```
module: ietf-keychain
  +--rw keychain
  |    +--rw private-keys
  |    |    +--rw private-key* [name]
  |    |    |    +--rw name                                    string
  |    |    |    +--ro algorithm?                              enumeration
  |    |    |    +--ro key-length?                             uint32
  |    |    |    +--ro public-key?                             string
  |    |    |    +--rw certificates
  |    |    |    |    +--rw certificate* [name]
  |    |    |    |    +--rw name string
  |    |    |    |    +--rw chain? binary
  |    |    |    +--x generate-certificate-signing-request
  |    |    |    |    +--w input
  |    |    |    |    |    +--w subject        binary
  |    |    |    |    |    +--w attributes? binary
  |    |    |    |    +--ro output
  |    |    |    |    |    +--ro certificate-signing-request binary
  |    +--x generate-private-key
  |    |    +--w input
  |    |    |    +--w name string
  |    |    |    +--w algorithm enumeration
  |    |    |    +--w key-length? uint32
  |    +--rw trusted-certificates* [name]
  |    |    +--rw name string
  |    |    +--rw description? string
  |    +--rw trusted-certificate* [name]
  |       +--rw name string
  |       +--rw certificate? binary
```

4.1.2. Example Usage

The following example illustrates the "generate-private-key" RPC in use with the RESTCONF protocol and JSON encoding.
The following example illustrates the action statement "generate-certificate-signing-request" action in use with the NETCONF protocol.

REQUEST
-------

```xml
<rpc message-id="101"
 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <action xmlns="urn:ietf:params:xml:ns:yang:1">
   <keychain xmlns="urn:ietf:params:xml:ns:yang:ietf-keychain">
     <private-keys>
       <private-key>
         <name>ex-key-sect571r1</name>
         <generate-certificate-signing-request>
           <subject>
             cztvaWRoc2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvO2R
             manZvO3NkZmJpdmhzZGZpbHVidjtvc2lkZmhidml1bHNlmo0
             Z2aXNiZGZpYmhzZG87ZmJvO3NkZ25iO29pLmR6Zgo=
           </subject>
           <attributes>
             bwtaKRo2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvut4
             arnZvO3NkZmJpdmhzZGZpbHVidjtvc2lkZmhidml1bHNkYm
           </attributes>
         </generate-certificate-signing-request>
       </private-key>
     </private-keys>
   </keychain>
 </action>
</rpc>
```
The following example illustrates what a fully configured keychain object might look like. The private-key shown below is consistent with the generate-private-key and generate-certificate-signing-request examples above. This example also assumes that the resulting CA-signed certificate has been configured back onto the server. Lastly, this example shows that three lists of trusted certificates having been configured.

```xml
<keychain xmlns="urn:ietf:params:xml:ns:yang:ietf-keychain">
  ...
</keychain>
```

Watsen & Schoenwaelder   Expires April 11, 2016                [Page 11]
<!-- private keys and associated certificates -->
<private-keys>
  <private-key>
    <name>ex-key-sect571r1</name>
    <algorithm>sect571r1</algorithm>
    <public-key>
      cztvaWRoc2ZmZtaHNNkZmdramRzZnZzZGtmmam5idNv02RmanZv03NkZmJpdmhzZGzpbHVidjtvC21kZmhidml1bHNNkYm2aX1ZG2pymhZGZ87ZmJvO3NkZ251O29pLmR62go=
    </public-key>
    <certificates>
      <certificate>
        <name>ex-key-sect571r1-cert</name>
        <data>
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z0F3SUJBZ01KQUpRT2t3bGpNK2pjTUEwR0NTcUdTSwizRFFQkJRUVFNRFFQ3pBkSnT1YQKZkFZVEFv51RNUkF3RgdZRZFRUUtFd2RsZUdGdGNHeGxNUk13RVPZFRZRUNRFd3BEVWt3Z1NYTnpkV1Z5U10 WaPe dIR1V4RxPBKJnT1ZCQ11UQ2tO1URDQjpm04xWHjd2dOHeDUEV1 KS29aShw2Y04KQQFQkJRQUrnWBtBUlHskFvR0JBTXVZmZmFNEV3ElI1QWMr1RsTkNmcOD6cEw1Um5ydXXsOFRIcU7tdGZQY3N02k1KT1FAN1nN1NW1dsMldzaHE1bUViCkJNnjItGNzdjbTAuU25fcFEO3nVbXBDt2YKQwInQKFBR2pnyX3Z2Frd0hWURWUjBPQkJZRUZKY1o2WUR1r0c3d4aj1pb3JtREdsRUNCVTFNR1FHQTQVFZApJdIJKuZU1QUZKYlo2WUR1R01NDB4aj1Pb3JtREdsRUNCVTFVY1rTmPBME1R3cmKMTUE0E0xVWMRFd0VCL3dRUF3SUNCREFTQmdOVXhSTUJBZjhFQ0RBR0FRSC9BZ0VBTEUwR0NTcUdTSWizRFFQgpCUUVBQ7RHQkJFMMmxrWmFNGwcyAR6MvNh2nZPbnBneHA4eG00SHRhb1adPHLazFLSF3BxTXp4YXJCbPSHilLck1Vbc9GVzRtV1RQSF1VDeEtFTE40NEY2zmk2dc4d0tSSElkY1W1LoGTMqLS0VXSTF4K111aD2mzczrGQ1QXg1R5f  SWHgzjvdMV2xZTgotLS0tLUVORCDRVJUSUZJQ0FURS0tLS0tCg==
        </data>
      </certificate>
    </certificates>
  </private-key>
</private-keys>

<!-- trusted netconf/restconf client certificates -->
<trusted-certificates>
  <name>explicitly-trusted-client-certs</name>
  <description>
    Specific client authentication certificates that are to be explicitly trusted NETCONF/RESTCONF clients. These are needed for client certificates not signed by our CA.
  </description>
  <trusted-certificate>
    <name>George Jetson</name>
    <certificate>
    </certificate>
  </trusted-certificate>
</trusted-certificates>
<trusted-certificates>
  <name>Fred Flinstone</name>
  <certificate>
    VlEV1FRREV3vm9ZWEJ3ZVDrQm5qU5C5z2txaGtpRz13MEJBuVUGVQFQm
    pRxqdWtDcm2ZRUE11RPFZwSs12p2d1XTW44eUh3hObUOrauUvYuV
    rUrpPQy9hSFA3eGJX0Wira1054zStUa2hrnBSL3ubvhsTjhS2Ud1ODhG
    NgeEkJ3U9cNvFJrVU1TUnBoVQFP0FSSFdSv0VPCk1CMEdeBMKbGd
    VEJiZ0JITEw1d1EmhlpRhV0tvwVHlLWw4cF3J121OYUICeERD05ER
    V6VQVZC05QKHFV11N0QJY0xN05DNUNVNRHBSN16UGz8REF
    NQmdOVkhSTUJB2jhfCkFqOQFQTRHFQTFVZ3RUVdIFQXdsJSQgDQnB
    Z05W5FI4RLqQmdNR2jSFXFZ2hOnW9kSFJ3T2k4dp1yaGgKylhCclpTN
    Wpi1MjB2WlhoaGJYQnNaUzvY215aU9LUTJNRF4Q3pBskJnTlzQV1UQW
    xWVE1SQxExE11E1FZwspF02Rs2UdGdGNH3xUk13RVPZFRZUURFd3B
    EWTzl31NYPnkVC125UeW0NtcUdTSWizREFFQkJRUXFBNdEcKFFC3BK
    WmdsK2gyTTg3QmtGJjWb1CdFFVAwC30EgrRkYyRTFwdSt4ZVRJbVFEM
    TQzcJS5k0MIPQlzV5eGUKN2QxMkxCV0dxUjUrbE15N01L21ka2M4al
    zSNwSDwVXBCYnA4dmnFzJma3RqZHBxeFppUUtBndWZTFZ7wot
    LS0tLUVORCDRVRVUZ3J0qaURS0tLS0t
  </certificate>
</trusted-certificate>
</trusted-certificates>
</trusted-certificates>

</trusted-certificates>
</trusted-certificates>
</trusted-certificates>
<!-- trust anchors for netconf/restconf clients -->
<trusted-certificates>
  <name>deployment-specific-ca-certs</name>
  <description>
    Trust anchors used only to authenticate NETCONF/RESTCONF
    client connections. Since our security policy only allows
    authentication for clients having a certificate signed by
    our CA, we only configure its certificate below.
  </description>
  <trusted-certificate>
    
  </trusted-certificate>
</trusted-certificates>
<description>
trusted-certificate

<name>ca.example.com</name>
<certificate>
WmdsK2gyTTg3QmtGMjWhW1CdFFVaWc3OEgrRkYyRTFwdsT4ZVRJbVFVM
1LQ1lsdWpOcJFTMnLR05EMUc20VJpK2FWMGr2NTdzNCtadhVJ0zpGRyj
zSFNwSDwvXBCYnA4dmtNaNFzjZma3RqZHBxeFppUUtBndWZTFZwot
NGcEk3UE90cNFSRjWuTUNBd0VBFQPQ0FSXdnZ0VFck1CMEdBMVkcRgd
VEJiZ0JWEd1bUEdMhplHRjYFVzcHkVvYHFLNw4cJFJB1Z0UYU0cERzd05ER
V6QVCJZ05WQkFNVENrT1NUQ0JYiZONMpvYSONUUNVRHBs116U6G8zREF
NQmdOVk5hSTUJB2jFckFqQUNFQTHQTFVZER3UIvd1FFQdJSGdEQnBA
Z05WF54r1Q0mdNRjZnSXFBZ2hoNW9kSFJ3T2K4dlpYagGyHc1pTN
Wp1MjB2W1hoaGJYqNaUzVqy215aU9LUTJNRF4Q3pBSkJnT1ZCQVIuQW
QmdOVkJNBWRbFzU7VBJdb0rnWURWUFLRdxbAplR0Z0Y0d4bERHndEQ
MKf6a3hjU1VWqprHR0dvS1UeUC1SR0Wm0vK3B0R2FieXDMjBRd2zkZ
25PZnpZNEhONApXY0pTaUZ2wkLytW3s3TRORUZXZ9S9Dp4NUL1XmdN2
RJZUQFRSTSCQg==
</certificate>
</trusted-certificate>
</trusted-certificates>

<!-- trust anchors for random HTTPS servers on Internet -->

trusted-certificates

<name>common-ca-certs</name>
<description>
Trusted certificates to authenticate common HTTPS servers. These
certificates are similar to those that might be shipped with a web
browser.
</description>
<trusted-certificate>
<name>ex-certificate-authority</name>
<certificate>
WmdsK2gyTTg3QmtGMjWhW1CdFFVaWc3OEgrRkYyRTFwdsT4ZVRJbVFVM
1LQ1lsdWpOcJFTMnLR05EMUc20VJpK2FWMGr2NTdzNCtadhVJ0zpGRyj
zSFNwSDwvXBCYnA4dmtNaNFzjZma3RqZHBxeFppUUtBndWZTFZwot
NGcEk3UE90cNFSRjWuTUNBd0VBFQPQ0FSXdnZ0VFck1CMEdBMVkcRgd
VEJiZ0JWEd1bUEdMhplHRjYFVzcHkVvYHFLNw4cJFJB1Z0UYU0cERzd05ER
V6QVCJZ05WQkFNVENrT1NUQ0JYiZONMpvYSONUUNVRHBs116U6G8zREF
NQmdOVk5hSTUJB2jFckFqQUNFQTHQTFVZER3UIvd1FFQdJSGdEQnBA
Z05WF54r1Q0mdNRjZnSXFBZ2hoNW9kSFJ3T2K4dlpYagGyHc1pTN
Wp1MjB2W1hoaGJYqNaUzVqy215aU9LUTJNRF4Q3pBSkJnT1ZCQVIuQW
QmdOVkJNBWRbFzU7VBJdb0rnWURWUFLRdxbAplR0Z0Y0d4bERHndEQ
MKf6a3hjU1VWqprHR0dvS1UeUC1SR0Wm0vK3B0R2FieXDMjBRd2zkZ
25PZnpZNEhONApXY0pTaUZ2wkLytW3s3TRORUZXZ9S9Dp4NUL1XmdN2
RJZUQFRSTSCQg==
</certificate>
</trusted-certificate>
</trusted-certificates>
4.1.3. YANG Model

<CODE BEGINS> file "ietf-keychain@2015-10-09.yang"

module ietf-keychain {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-keychain";
  prefix "kc";

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

  contact
    "WG Web: <http://tools.ietf.org/wg/netconf/>
    WG List: <mailto:netconf@ietf.org>
    WG Chair: Mehmet Ersue
      <mailto:mehmet.ersue@nsn.com>
    WG Chair: Mahesh Jethanandani
      <mailto:mjethanandani@gmail.com>
    Editor: Kent Watsen
      <mailto:kwatsen@juniper.net>";

  description
    "This module defines a keychain to centralize management of
    security credentials."

  Copyright (c) 2014 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 VVVV; see
  the RFC itself for full legal notices."

  revision "2015-10-09" {
container keychain {
    description
    "A list of private-keys and their associated certificates, as well as lists of trusted certificates for client certificate authentication. RPCs are provided to generate a new private key and to generate a certificate signing requests.";
}

container private-keys {
    description
    "A list of private key maintained by the keychain.";
    list private-key {
        key name;
        description
        "A private key.";
        leaf name {
            type string;
            description
            "An arbitrary name for the private key.";
        }
        leaf algorithm {
            type enumeration {
                enum rsa { description "TBD"; }
                enum dsa { description "TBD"; }
                enum secp192r1 { description "TBD"; }
                enum secp193k1 { description "TBD"; }
                enum sect163k1 { description "TBD"; }
                enum sect163r2 { description "TBD"; }
                enum secp224r1 { description "TBD"; }
                enum sect233k1 { description "TBD"; }
                enum sect233r1 { description "TBD"; }
                enum secp256r1 { description "TBD"; }
                enum sect283k1 { description "TBD"; }
                enum sect283r1 { description "TBD"; }
                enum secp384r1 { description "TBD"; }
                enum sect409k1 { description "TBD"; }
                enum sect409r1 { description "TBD"; }
                enum secp521r1 { description "TBD"; }
                enum sect571k1 { description "TBD"; }
                enum sect571r1 { description "TBD"; }
            }
            config false;
            description
leaf key-length {
    type uint32;
    config false;
    description "The key-length used by the private key.";
}
leaf public-key {
    type string;
    config false;
    description "The public-key matching the private key.";
}
container certificates {
    list certificate {
        key name;
        description "A certificate for this public key.";
        leaf name {
            type string;
            description "An arbitrary name for the certificate.";
        }
        leaf chain {
            type binary;
            description "The certificate itself, as well as an ordered sequence of intermediate certificates leading to a trust anchor, as specified by RFC 5246, Section 7.4.2.";
        }
    }
}
action generate-certificate-signing-request {
    description "Generates a certificate signing request structure for the associated private key using the passed subject and attribute values.";
    input {
        leaf subject {
            type binary;
            mandatory true;
        }
    }
}
description
"The distinguished name of the certificate subject
(the entity whose public key is to be certified). This field is encoded the same as the ‘subject’
field in the CertificationRequestInfo type defined in RFC 2986, Section 4.1.";
reference
"RFC 2986: PKCS #10: Certification Request Syntax
Specification Version 1.7";

leaf attributes {
  type binary;
  description
  "A collection of attributes providing additional
  information about the subject of the certificate.
  This field is encoded the same as the ‘attributes’
  field in the CertificationRequestInfo type defined
  in RFC 2986, Section 4.1.";
  reference
  "RFC 2986: PKCS #10: Certification Request Syntax
  Specification Version 1.7";
}

output {
  leaf certificate-signing-request {
    type binary;
    mandatory true;
    description
    "The certificate signing request to be signed by
    a certificate authority. This field is encoded
    as the CertificationRequest type defined in
    RFC 2986, Section 4.2.";
    reference
    "RFC 2986: PKCS #10: Certification Request Syntax
    Specification Version 1.7";
  }
}

action generate-private-key {
  description
  "Generates a private key using the specified algorithm and
  key length.";
  input {
    leaf name {
      type string;
      mandatory true;
      description
      "The name of the certificate authority.
      This field is encoded as the
      CertificateRequestInfo type defined
      in RFC 2986, Section 4.2.";
      reference
      "RFC 2986: PKCS #10: Certification Request Syntax
      Specification Version 1.7";
    }
  }
}
"The name this private-key should have when listed in /keychain/private-keys. As such, the passed value must not match any existing 'name' value."
}
leaf algorithm {
  type enumeration {
    enum rsa { description "TBD"; }
    enum dsa { description "TBD"; }
    enum secp192r1 { description "TBD"; }
    enum sect163k1 { description "TBD"; }
    enum sect163r2 { description "TBD"; }
    enum secp224r1 { description "TBD"; }
    enum sect233k1 { description "TBD"; }
    enum sect233r1 { description "TBD"; }
    enum secp256r1 { description "TBD"; }
    enum sect283k1 { description "TBD"; }
    enum sect283r1 { description "TBD"; }
    enum secp384r1 { description "TBD"; }
    enum sect409k1 { description "TBD"; }
    enum sect409r1 { description "TBD"; }
    enum secp521r1 { description "TBD"; }
    enum sect571k1 { description "TBD"; }
    enum sect571r1 { description "TBD"; }
  }
  mandatory true;
  description
    "The algorithm to be used.";
}
leaf key-length {
  type uint32;
  description
    "For algorithms that need a key length specified when generating the key."
}
)
)
list trusted-certificates {
  key name;
  description
    "A list of lists of trusted certificates.";
  leaf name {
    type string;
    description
      "An arbitrary name for this list of trusted certificates.";
  }
}
4.2. The SSH Server Model

The SSH Server model presented in this section presents two YANG groupings, one for a server that opens a socket to accept TCP connections on, and another for a server that has had the TCP connection opened for it already (e.g., inetd).

The SSH Server model (like the TLS Server model presented below) is provided as a grouping so that it can be used in different contexts. For instance, the NETCONF Server model presented in Section 4.4 uses one grouping to configure a NETCONF server listening for connections and the other grouping to configure NETCONF call home.

A shared characteristic between both groupings is the ability to configure which host key is presented to clients, the private key for
which is held in the keychain configuration presented before. Another shared characteristic is the ability to configure which trusted CA or client certificates the server should be used to authenticate clients when using X.509 based client certificates [RFC6187].

4.2.1. Tree Diagram

The following tree diagram represents the data model for the grouping used to configure an SSH server to listen for TCP connections. The tree diagram for the other grouping is not provided, but it is the same except without the "address" and "port" fields.

NOTE: the diagram below shows "listening-ssh-server" as a YANG container (not a grouping). This temporary container was created only to enable the `pyang` tool to output the tree diagram, as groupings by themselves have no protocol accessible nodes, and hence 'pyang' would output an empty tree diagram.

```Yang
module: ietf-ssh-server
  +--rw listening-ssh-server
    +--rw address?            inet:ip-address
    +--rw port                inet:port-number
    +--rw host-keys
      +--rw host-key* [name]
        +--rw name           string
        +--rw (type)?
          +--:(public-key)
            +--rw public-key?    -> /kc:keychain/private-keys/private-key/name
          |                     +--:(certificate)
            +--rw certificate?   -> /kc:keychain/private-keys/private-key/certificates/certificate/name {ssh-x509-certs}
    +--rw client-cert-auth {ssh-x509-certs}?
    +--rw trusted-ca-certs?       -> /kc:keychain/trusted-certificates/name
    +--rw trusted-client-certs?   -> /kc:keychain/trusted-certificates/name
```

4.2.2. Example Usage

This section shows how it would appear if the temporary listening-ssh-server container just mentioned above were populated with some data. This example is consistent with the examples presented earlier in this document.
<listening-ssh-server
xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-server">
  <port>830</port>
  <host-keys>
    <host-key>
      <name>deployment-specific-certificate</name>
      <certificate>ex-key-sect571r1-cert</certificate>
    </host-key>
  </host-keys>
</listening-ssh-server>

4.2.3. YANG Model

<CODE BEGINS> file "ietf-ssh-server@2015-10-09.yang"

module ietf-ssh-server {  
yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-ssh-server";
  prefix "ts";

  import ietf-inet-types {           // RFC 6991
    prefix inet;
  }
  import ietf-keychain {           // RFC VVVV
    prefix kc;
    revision-date 2015-10-09;
  }

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

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

    WG Chair: Mehmet Ersue
      <mailto:mehmet.ersue@nsn.com>"
This module defines a reusable grouping for a SSH server that can be used as a basis for specific SSH server instances.

Copyright (c) 2014 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 VVVV; see the RFC itself for full legal notices.

revision "2015-10-09" {
  description
    "Initial version";
  reference
    "RFC VVVV: NETCONF Server and RESTCONF Server Configuration Models";
}

// features
feature ssh-x509-certs {
  description
    "The ssh-x509-certs feature indicates that the NETCONF server supports RFC 6187";
  reference
    "RFC 6187: X.509v3 Certificates for Secure Shell Authentication";
}

// grouping
grouping non-listening-ssh-server-grouping {
  description
    "A reusable grouping for a SSH server that can be used as a basis for specific SSH server instances.";
}
container host-keys {
  description
    "The list of host-keys the SSH server will present when
    establishing a SSH connection.";
  list host-key {
    key name;
    min-elements 1;
    ordered-by user;
    description
      "An ordered list of host keys the SSH server advertises
      when sending its ??? message.";
    reference
      "RFC ????: ...";
    leaf name {
      type string;
      mandatory true;
      description
        "An arbitrary name for this host-key";
    }
  choice type {
    description
      "The type of host key being specified";
    leaf public-key {
      type leafref {
        path "/kc:keychain/kc:private-keys/kc:private-key/"
        + "kc:name";
      }
      description
        "The name of a private-key in the keychain.";
    }
    leaf certificate {
      if-feature ssh-x509-certs;
      type leafref {
        path "/kc:certificate/kc:certificate/kc:name";
      }
      description
        "The name of a certificate in the keychain.";
    }
  }
}
}

container client-cert-auth {
  if-feature ssh-x509-certs;
  description
    "A reference to a list of trusted certificate authority (CA)
    certificates and a reference to a list of trusted client
leaf trusted-ca-certs {
    type leafref {
        path "/kc:keychain/kc:trusted-certificates/kc:name";
    }
    description
        "A reference to a list of certificate authority (CA) certificates used by the SSH server to authenticate SSH client certificates.";
}

leaf trusted-client-certs {
    type leafref {
        path "/kc:keychain/kc:trusted-certificates/kc:name";
    }
    description
        "A reference to a list of client certificates used by the SSH server to authenticate SSH client certificates. A clients certificate is authenticated if it is an exact match to a configured trusted client certificate.";
}
}
}

grouping listening-ssh-server-grouping {
    description
        "A reusable grouping for a SSH server that can be used as a basis for specific SSH server instances.";
    leaf address {
        type inet:ip-address;
        description
            "The IP address of the interface to listen on. The SSH server will listen on all interfaces if no value is specified.";
    }
    leaf port {
        type inet:port-number;
        mandatory true;  // will a default augmented in work?
        description
            "The local port number on this interface the SSH server listens on.";
    }
    uses non-listening-ssh-server-grouping;
}

// RFC Editor: please remove the following container block
// when publishing this document as an RFC.
container listening-ssh-server {
    description
        "This container is only present to enable 'pyang'
        tree diagram output, as a grouping by itself has
        no protocol accessible nodes to output."
    uses listening-ssh-server-grouping;
}

<CODE ENDS>

4.3. The TLS Server Model

The TLS Server model presented in this section presents two YANG
groupings, one for a server that opens a socket to accept TCP
connections on, and another for a server that has had the TCP
connection opened for it already (e.g., inetd).

The TLS Server model (like the SSH Server model presented above) is
provided as a grouping so that it can be used in different contexts.
For instance, the NETCONF Server model presented in Section 4.4 uses
one grouping to configure a NETCONF server listening for connections
and the other grouping to configure NETCONF call home.

A shared characteristic between both groupings is the ability to
configure which server certificate is presented to clients, the
private key for which is held in the keychain model presented in
Section 4.1. Another shared characteristic is the ability to
configure which trusted CA or client certificates the server should
be used to authenticate clients.

4.3.1. Tree Diagram

The following tree diagram represents the data model for the grouping
used to configure an TLS server to listen for TCP connections. The
tree diagram for the other grouping is not provided, but it is the
same except without the "address" and "port" fields.

NOTE: the diagram below shows "listening-ssh-server" as a YANG
container (not a grouping). This temporary container was created
only to enable the 'pyang' tool to output the tree diagram, as
groupings by themselves have no protocol accessible nodes, and hence
'pyang' would output an empty tree diagram.
module: ietf-tls-server
  +--rw listening-tls-server
    +--rw address? inet:ip-address
    +--rw port inet:port-number
    +--rw certificates
      +--rw certificate* [name]
        +--rw name -> /kc:keychain/private-keys/private-key/certificates/certificate/name
        +--rw trusted-ca-certs? -> /kc:keychain/trusted-certificates/name
        +--rw trusted-client-certs? -> /kc:keychain/trusted-certificates/name

4.3.2. Example Usage

<listening-tls-server
 xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-server">
  <port>6513</port>
  <certificates>
    <certificate>
      <name>ex-key-sect571r1-cert</name>
    </certificate>
  </certificates>
  <client-auth>
    <trusted-ca-certs>
      deployment-specific-ca-certs
    </trusted-ca-certs>
    <trusted-client-certs>
      explicitly-trusted-client-certs
    </trusted-client-certs>
  </client-auth>
</listening-tls-server>

4.3.3. YANG Model

<CODE BEGINS> file "ietf-tls-server@2015-10-09.yang"

module ietf-tls-server {  
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-tls-server";
  prefix "ts";

  import ietf-inet-types {  // RFC 6991
    prefix inet;
  }

Watsen & Schoenwaelder   Expires April 11, 2016                [Page 27]
import ietf-keychain {  
    prefix kc;                      // RFC VVVV  
    revision-date 2015-10-09;  
}  

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

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

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

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

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

description  
"This module defines a reusable grouping for a TLS server that  
can be used as a basis for specific TLS server instances.  

Copyright (c) 2014 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 VVVV; see  
the RFC itself for full legal notices.";

revision "2015-10-09" {  
    description  
    "Initial version";  
    reference  
    "RFC VVVV: NETCONF Server and RESTCONF Server Configuration  
    Models";  
}
// grouping
 grouping non-listening-tls-server-grouping {
     description
      "A reusable grouping for a TLS server that can be used as a
      basis for specific TLS server instances.";
     container certificates {
          description
            "The list of certificates the TLS server will present when
            establishing a TLS connection.";
          list certificate {
              key name;
              min-elements 1;
              description
                "An unordered list of certificates the TLS server can pick
                from when sending its Server Certificate message.";
              reference
                "RFC 5246: The TLS Protocol, Section 7.4.2";
              leaf name {
                  type leafref {
                      path "/kc:keychain/kc:private-keys/kc:private-key/
                      + "kc:certificates/kc:certificate/kc:name";
                  }
                  description
                    "The name of the certificate in the keychain.";
              }
          }
     }
     container client-auth {
          description
            "A reference to a list of trusted certificate authority (CA)
            certificates and a reference to a list of trusted client
            certificates.";
          leaf trusted-ca-certs {
              type leafref {
                  path "/kc:keychain/kc:trusted-certificates/kc:name";
              }
              description
                "A reference to a list of certificate authority (CA)
                certificates used by the TLS server to authenticate
                TLS client certificates.";
          }
          leaf trusted-client-certs {
              type leafref {
                  path "/kc:keychain/kc:trusted-certificates/kc:name";
              }
              description
"A reference to a list of client certificates used by the TLS server to authenticate TLS client certificates. A client certificate is authenticated if it is an exact match to a configured trusted client certificate."
}
)
}
}

// RFC Editor: please remove the following container block when publishing this document as an RFC.
// container listening-tls-server {
//    description
//        "This container is only present to enable 'pyang' tree diagram output, as a grouping by itself has no protocol accessible nodes to output."
//        uses listening-tls-server-grouping;
// }
}
4.4. The NETCONF Server Model

The NETCONF Server model presented in this section supports servers both listening for connections to accept as well as initiating call-home connections. This model also supports both the SSH and TLS transport protocols, using the SSH Server and TLS Server groupings presented in Section 4.2 and Section 4.3 respectively. All private keys and trusted certificates are held in the keychain model presented in Section 4.1. YANG feature statements are used to enable implementations to advertise which parts of the model the NETCONF server supports.

4.4.1. Tree Diagram

The following tree diagram uses line-wrapping in order to comply with xml2rfc validation. This is annoying as I find that drafts (even txt drafts) look just fine with long lines – maybe xml2rfc should remove this warning? – or pyang could have an option to suppress printing leafref paths?

```
module: ietf-netconf-server
  +--rw netconf-server
      +--rw session-options
          |  +--rw hello-timeout? uint16
          +--rw listen *((ssh-listen or tls-listen))?
              |  +--rw max-sessions? uint16
              +--rw idle-timeout? uint16
              +--rw endpoint* [name]
                  +--rw name string
                  +--rw (transport)
                      +--:(ssh) {ssh-listen}?
                          +--rw ssh
                              |  +--rw address? inet:ip-address
                              +--rw port inet:port-number
                              +--rw host-keys
                                  |  +--rw host-key* [name]
                                  |      +--rw name string
                                  |      +--rw (type)?
                                  |          +--:(public-key)
                                  |              +--rw public-key? -> /kc:keychain/private-keys/private-key/name
                              +--rw client-cert-auth {ssh-x509-certs}?
                                  +--rw trusted-ca-certs? -> /kc:keychain/private-keys/private-key/certificates/certificate/name {ssh-x509-certs}?
```

Watsen & Schoenwaelder   Expires April 11, 2016                [Page 31]
rusted-certificates/name
  +--rw trusted-client-certs?  -> /kc:keychain/t
rusted-certificates/name
  +--:(tls) {tls-listen}?
    +--rw tls
      +--rw address?   inet:ip-address
      +--rw port       inet:port-number
      +--rw certificates
        +--rw certificate* [name]
          +--rw name    -> /kc:keychain/private-keys/p
private-key/certificates/certificate/name
  +--rw trusted-ca-certs?  -> /kc:keychain/t
rusted-certificates/name
  +--rw trusted-client-certs?  -> /kc:keychain/t
rusted-certificates/name
  +--rw cert-maps
    +--rw cert-to-name* [id]
      +--rw id       uint32
      +--rw fingerprint  x509c2n:tls-fingerprint
rusted-certificates/name
  +--rw call-home { (ssh-call-home or tls-call-home) }?
    +--rw netconf-client* [name]
      +--rw name          string
      +--rw (transport)
        +--:(ssh) {ssh-call-home}?
          +--rw ssh
            +--rw endpoints
              +--rw endpoint* [name]
                +--rw name           string
                +--rw address        inet:host
                +--rw port?          inet:port-number
              +--rw host-keys
                +--rw host-key* [name]
                  +--rw name           string
                  +--rw (type)?
                    +--:(public-key)
                      +--rw public-key?  -> /kc:keychain/p
private-keys/private-key/name
  +--:(certificate)
    +--rw certificate?  -> /kc:keychain/p
private-keys/private-key/certificates/certificate/name { (ssh-x509-certs) }?
  +--rw client-cert-auth {ssh-x509-certs}?
  +--rw trusted-ca-certs?  -> /kc:keychain/t
rusted-certificates/name
  +--rw trusted-client-certs?  -> /kc:keychain/t
4.4.2. Example Usage

Configuring a NETCONF Server to listen for NETCONF client connections using both the SSH and TLS transport protocols, as well as configuring call-home to two NETCONF clients, one using SSH and the other using TLS.
This example is consistent with other examples presented in this document.

```xml
<netconf-server
 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
 <listen>
   <!-- listening for SSH connections -->
   <endpoint>
     <name>netconf/ssh</name>
     <ssh>
       <host-keys>
         <host-key>
           <public-key>my-rsa-key</public-key>
         </host-key>
         <host-key>
           <certificate>TPM key</certificate>
         </host-key>
         <host-keys>
           <client-cert-auth>
             <trusted-ca-certs>
               <trusteed-certs>
                 <deployment-specific-ca-certs>
                 </trusted-ca-certs>
                 <trusted-client-certs>
                   <explicitly-trusted-client-certs>
                 </trusted-client-certs>
               </trusted-client-certs>
             </trusted-ca-certs>
           </client-cert-auth>
         </host-keys>
       </host-key>
     </ssh>
   </endpoint>

   <!-- listening for TLS connections -->
   <endpoint>
     <name>netconf/tls</name>
     <tls>
       <address>11.22.33.44</address>
       <certificates>
         <certificate>ex-key-sect571r1-cert</certificate>
       </certificates>
       <client-auth>
         <trusted-ca-certs>
           <trusteed-certs>
             <deployment-specific-ca-certs>
           </trusted-ca-certs>
           <trusted-client-certs>
             <explicitly-trusted-client-certs>
           </trusted-client-certs>
         </trusted-client-certs>
       </client-auth>
     </tls>
   </endpoint>
</listen>
```
<id>1</id>
<fingerprint>11:0A:05:11:00</fingerprint>
<map-type>x509c2n:san-any</map-type>
</cert-to-name>
<cert-to-name>
<id>2</id>
<fingerprint>B3:4F:A1:8C:54</fingerprint>
<map-type>x509c2n:specified</map-type>
<name>scooby-doo</name>
</cert-to-name>
</cert-maps>
</client-auth>
</tls>
</endpoint>
</listen>
</call-home>

<!-- calling home to an SSH-based NETCONF client -->
<netconf-client>
 <name>config-mgr</name>
 <ssh>
  <endpoints>
   <endpoint>
    <name>east-data-center</name>
    <address>11.22.33.44</address>
   </endpoint>
   <endpoint>
    <name>west-data-center</name>
    <address>55.66.77.88</address>
   </endpoint>
  </endpoints>
  <host-keys>
   <host-key>
    <certificate>TPM key</certificate>
   </host-key>
  </host-keys>
  <client-cert-auth>
   <trusted-ca-certs>
    deployment-specific-ca-certs
   </trusted-ca-certs>
   <trusted-client-certs>
    explicitly-trusted-client-certs
   </trusted-client-certs>
  </client-cert-auth>
 </ssh>
 <connection-type>
  <periodic>
<idle-timeout>300</idle-timeout>
<reconnect-timeout>60</reconnect-timeout>
</periodic>
</connection-type>
<reconnect-strategy>
<start-with>last-connected</start-with>
<max-attempts>3</max-attempts>
</reconnect-strategy>
</netconf-client>

<!-- calling home to a TLS-based NETCONF client -->
<netconf-client>
  <name>event-correlator</name>
  <tls>
    <endpoints>
      <endpoint>
        <name>east-data-center</name>
        <address>22.33.44.55</address>
      </endpoint>
      <endpoint>
        <name>west-data-center</name>
        <address>33.44.55.66</address>
      </endpoint>
    </endpoints>
    <certificates>
      <certificate>ex-key-sect571r1-cert</certificate>
    </certificates>
    <client-auth>
      <trusted-ca-certs>
        deployment-specific-ca-certs
      </trusted-ca-certs>
      <trusted-client-certs>
        explicitly-trusted-client-certs
      </trusted-client-certs>
      <cert-maps>
        <cert-to-name>
          <id>1</id>
          <fingerprint>11:0A:05:11:00</fingerprint>
          <map-type>x509c2n:san-any</map-type>
        </cert-to-name>
        <cert-to-name>
          <id>2</id>
          <fingerprint>B3:4F:A1:8C:54</fingerprint>
          <map-type>x509c2n:specified</map-type>
          <name>scooby-doo</name>
        </cert-to-name>
      </cert-maps>
    </client-auth>
  </tls>
</netconf-client>
4.4.3. YANG Model

This YANG module imports YANG types from [RFC6991] and [RFC7407].

<CODE BEGINS> file "ietf-netconf-server@2015-10-09.yang"

module ietf-netconf-server {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-server";
  prefix "ncserver";

  import ietf-inet-types { // RFC 6991
    prefix inet;
  }
  import ietf-x509-cert-to-name { // RFC 7407
    prefix x509c2n;
  }
  import ietf-ssh-server { // RFC VVVV
    prefix ss;
    revision-date 2015-10-09;
  }
  import ietf-tls-server { // RFC VVVV
    prefix ts;
    revision-date 2015-10-09;
  }
}
This module contains a collection of YANG definitions for configuring NETCONF servers.

Copyright (c) 2014 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 VVVV; see the RFC itself for full legal notices."

revision "2015-10-09" {
  description "Initial version";
  reference "RFC VVVV: NETCONF Server and RESTCONF Server Configuration Models";
}

// Features

feature ssh-listen {
  description "The ssh-listen feature indicates that the NETCONF server
supports opening a port to accept NETCONF over SSH client connections.
reference
"RFC 6242: Using the NETCONF Protocol over Secure Shell (SSH)";
}

feature ssh-call-home {
  description
  "The ssh-call-home feature indicates that the NETCONF server supports
   initiating a NETCONF over SSH call home connection to NETCONF clients."
  reference
  "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
}

feature tls-listen {
  description
  "The tls-listen feature indicates that the NETCONF server supports
   opening a port to accept NETCONF over TLS client connections.";
  reference
  "RFC 5539: Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication";
}

feature tls-call-home {
  description
  "The tls-call-home feature indicates that the NETCONF server supports
   initiating a NETCONF over TLS call home connection to NETCONF clients.";
  reference
  "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
}

feature ssh-x509-certs {
  description
  "The ssh-x509-certs feature indicates that the NETCONF server supports
   RFC 6187";
  reference
  "RFC 6187: X.509v3 Certificates for Secure Shell Authentication";
}

// top-level container (groupings below)
container netconf-server {
  description


"Top-level container for NETCONF server configuration."

container session-options {  // SHOULD WE REMOVE THIS ALTOGETHER?
    description
        "NETCONF session options, independent of transport
         or connection strategy.";
    leaf hello-timeout {
        type uint16;
        units "seconds";
        default 600;
        description
            "Specifies the maximum number of seconds that a SSH/TLS
             connection may wait for a hello message to be received.
             A connection will be dropped if no hello message is
             received before this number of seconds elapses. If set
to zero, then the server will wait forever for a hello
             message.";
    }
}

container listen {
    if-feature "(ssh-listen or tls-listen)";
    description
        "Configures listen behavior";
    leaf max-sessions {
        type uint16;
        default 0;
        description
            "Specifies the maximum number of concurrent sessions
             that can be active at one time. The value 0 indicates
             that no artificial session limit should be used.";
    }
    leaf idle-timeout {
        type uint16;
        units "seconds";
        default 3600; // one hour
        description
            "Specifies the maximum number of seconds that a NETCONF
             session may remain idle. A NETCONF session will be dropped
             if it is idle for an interval longer than this number of
             seconds. If set to zero, then the server will never drop
             a session because it is idle. Sessions that have a
             notification subscription active are never dropped.";
    }
    list endpoint {
        key name;
        description
            "List of endpoints to listen for NETCONF connections on.";
    }
leaf name {
    type string;
    description
        "An arbitrary name for the NETCONF listen endpoint.";
}
choice transport {
    mandatory true;
    description
        "Selects between available transports.";
    case ssh {
        if-feature ssh-listen;
        container ssh {
            description
                "SSH-specific listening configuration for inbound
                connections.";
            uses ss:listening-ssh-server-grouping {
                refine port {
                    default 830;
                }
            }
        }
    }
    case tls {
        if-feature tls-listen;
        container tls {
            description
                "TLS-specific listening configuration for inbound
                connections.";
            uses ts:listening-tls-server-grouping {
                refine port {
                    default 6513;
                }
            }
            augment "client-auth" {
                description
                    "Augments in the cert-to-name structure.";
                uses cert-maps-grouping;
            }
        }
    }
}
container call-home {
    if-feature "((ssh-call-home or tls-call-home))";
    description
        "Configures call-home behavior";
}
list netconf-client {
  key name;
  description
    "List of NETCONF clients the NETCONF server is to initiate
     call-home connections to."
  leaf name {
    type string;
    description
      "An arbitrary name for the remote NETCONF client."
  }
  choice transport {
    mandatory true;
    description
      "Selects between available transports."
    case ssh {
      if-feature ssh-call-home;
      container ssh {
        description
          "Specifies SSH-specific call-home transport
           configuration."
        uses endpoints-container {
          refine endpoints/endpoint/port {
            default 7777;
          }
        }
        uses ss:non-listening-ssh-server-grouping;
      }
    }
    case tls {
      if-feature tls-call-home;
      container tls {
        description
          "Specifies TLS-specific call-home transport
           configuration."
        uses endpoints-container {
          refine endpoints/endpoint/port {
            default 8888;
          }
        }
        uses ts:non-listening-tls-server-grouping {
          augment "client-auth" {
            description
              "Augments in the cert-to-name structure."
            uses cert-maps-grouping;
          }
        }
      }
    }
  }
}
container connection-type {
    description "Indicates the kind of connection to use.";
    choice connection-type {
        description "Selects between available connection types.";
        case persistent-connection {
            container persistent {
                presence true;
                description "Maintain a persistent connection to the NETCONF client. If the connection goes down, immediately start trying to reconnect to it, using the reconnection strategy.

This connection type minimizes any NETCONF client to NETCONF server data-transfer delay, albeit at the expense of holding resources longer.";
            }
        }
    }
    leaf idle-timeout {
        type uint32;
        units "seconds";
        default 86400; // one day;
        description "Specifies the maximum number of seconds that a NETCONF session may remain idle. A NETCONF session will be dropped if it is idle for an interval longer than this number of seconds. If set to zero, then the server will never drop a session because it is idle. Sessions that have a notification subscription active are never dropped.";
    }
}

container keep-alives {
    description "Configures the keep-alive policy, to proactively test the aliveness of the SSH/TLS client. An unresponsive SSH/TLS client will be dropped after approximately max-attempts * max-wait seconds.";
    reference "RFC YYY: NETCONF Call Home and RESTCONF Call Home, Section 3.1, item S6";
    leaf max-wait {
        type uint16 {
            range "1..max";
        }
        units seconds;
        default 30;
    }
description
  "Sets the amount of time in seconds after which
  if no data has been received from the SSH/TLS
  client, a SSH/TLS-level message will be sent
to test the aliveness of the SSH/TLS client.";}
leaf max-attempts {
  type uint8;
  default 3;
  description
  "Sets the number of maximum number of sequential
  keep-alive messages that can fail to obtain a
  response from the SSH/TLS client before assuming
  the SSH/TLS client is no longer alive.";
}

case periodic-connection {
  container periodic {
    presence true;
    description
    "Periodically connect to the NETCONF client, so that
    the NETCONF client may deliver messages pending for
    the NETCONF server. The NETCONF client is expected
to close the connection when it is ready to release
it, thus starting the NETCONF server’s timer until
next connection.";
  leaf idle-timeout {
    type uint16;
    units "seconds";
    default 300; // five minutes
    description
    "Specifies the maximum number of seconds that a
    NETCONF session may remain idle. A NETCONF
    session will be dropped if it is idle for an
    interval longer than this number of seconds.
    If set to zero, then the server will never drop
    a session because it is idle. Sessions that
    have a notification subscription active are
    never dropped.";
  }
  leaf reconnect_timeout {
    type uint16 {
      range "1..max";
    }
    units minutes;
    default 60;
description
"Sets the maximum amount of unconnected time the NETCONF server will wait before re-establishing a connection to the NETCONF client. The NETCONF server may initiate a connection before this time if desired (e.g., to deliver an event notification message).";
}
}
}
}
}
}
}
container reconnect-strategy {
  description
  "The reconnection strategy guides how a NETCONF server reconnects to a NETCONF client, after discovering its connection to the client has dropped. The NETCONF server starts with the specified endpoint and tries to connect to it max-attempts times before trying the next endpoint in the list (round robin).";
  leaf start-with {
    type enumeration {
      enum first-listed {
        description
        "Indicates that reconnections should start with the first endpoint listed.";
      }
      enum last-connected {
        description
        "Indicates that reconnections should start with the endpoint last connected to. If no previous connection has ever been established, then the first endpoint configured is used. NETCONF servers SHOULD be able to remember the last endpoint connected to across reboots.";
      }
    }
    default first-listed;
  }
  description
  "Specifies which of the NETCONF client’s endpoints the NETCONF server should start with when trying to connect to the NETCONF client.";
}
leaf max-attempts {
  type uint8 {
    range "1..max";
  }
  default 3;
description
"Specifies the number times the NETCONF server tries to
connect to a specific endpoint before moving on to the
next endpoint in the list (round robin).";
}
}
}

grouping cert-maps-grouping {
  description
  "A grouping that defines a container around the
cert-to-name structure defined in RFC 7407."
  container cert-maps {
    uses x509c2n:cert-to-name;
    description
    "The cert-maps container is used by a TLS-based NETCONF
server to map the NETCONF client’s presented X.509
certificate to a NETCONF username. If no matching and
valid cert-to-name list entry can be found, then the
NETCONF server MUST close the connection, and MUST NOT
accept NETCONF messages over it.";
    reference
    "RFC WWWW: NETCONF over TLS, Section 7";
  }
}

grouping endpoints-container {
  description
  "This grouping is used by both the ssh and tls containers
for call-home configurations.";
  container endpoints {
    description
    "Container for the list of endpoints.";
    list endpoint {
      key name;
      min-elements 1;
      ordered-by user;
      description
      "User-ordered list of endpoints for this NETCONF client.
Defining more than one enables high-availability.";
      leaf name {
        type string;
        description
        "An arbitrary name for this endpoint.";
      }
    }
  }
}
4.5. The RESTCONF Server Model

The RESTCONF Server model presented in this section supports servers both listening for connections to accept as well as initiating call-home connections. This model supports the TLS transport only, as RESTCONF only supports HTTPS, using the TLS Server groupings presented in Section 4.3. All private keys and trusted certificates are held in the keychain model presented in Section 4.1. YANG feature statements are used to enable implementations to advertise which parts of the model the RESTCONF server supports.

4.5.1. Tree Diagram

The following tree diagram uses line-wrapping in order to comply with xml2rfc validation. This is annoying as I find that drafts (even txt drafts) look just fine with long lines - maybe xml2rfc should remove this warning? - or pyang could have an option to suppress printing leafref paths?

module: ietf-restconf-server
   +--rw restconf-server
+++rw listen {tls-listen}?
    +++rw max-sessions? uint16
    +++rw endpoint* [name]
       +++rw name string
    +++rw (transport)
       +++:(tls) {tls-listen}?
          +++rw tls
             +++rw address? inet:ip-address
             +++rw port inet:port-number
             +++rw certificates
                +++rw certificate* [name]
                   +++rw name -> /kc:keychain/private-keys/private-key/certificates/certificate/name
             
    private-key/certificates/certificate/name
    +++rw client-auth
       +++rw trusted-ca-certs? -> /kc:keychain/trusted-certificates/name
       +++rw trusted-client-certs? -> /kc:keychain/trusted-certificates/name
    
    +++rw call-home {tls-call-home}? 
    +++rw restconf-client* [name]
       +++rw name string
    +++rw (transport)
       +++:(tls) {tls-call-home}?
          +++rw tls
             +++rw endpoints
                +++rw endpoint* [name]
                   +++rw name string
                   +++rw address inet:host
                   +++rw port? inet:port-number
                +++rw certificates
                   +++rw certificate* [name]
                      +++rw name -> /kc:keychain/private-keys/private-key/certificates/certificate/name
                private-key/certificates/certificate/name
                +++rw client-auth
                   +++rw trusted-ca-certs? -> /kc:keychain/trusted-certificates/name
                   +++rw trusted-client-certs? -> /kc:keychain/trusted-certificates/name
                
                +++rw cert-maps
                   +++rw cert-to-name* [id]
                      +++rw id uint32
                      +++rw fingerprint x509c2n:tls-fingerprint

    watsen & schoenwaelder   expires april 11, 2016   [page 48]
4.5.2. Example Usage

Configuring a RESTCONF Server to listen for RESTCONF client connections, as well as configuring call-home to one RESTCONF client.

This example is consistent with other examples presented in this document.

```xml
<restconf-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-server">

<!-- listening for TLS (HTTPS) connections -->
<listen>
  <endpoint>
    <name>netconf/tls</name>
    <tls>
      <address>11.22.33.44</address>
      <certificates>
        <certificate>ex-key-sect571r1-cert</certificate>
      </certificates>
      <trusted-ca-certs>
        deployment-specific-ca-certs
      </trusted-ca-certs>
      <trusted-client-certs>
        explicitly-trusted-client-certs
      </trusted-client-certs>
    </tls>
  </endpoint>
</listen>
```

Watsen & Schoenwaelder   Expires April 11, 2016   [Page 49]
<cert-to-name>
  <id>2</id>
  <fingerprint>B3:4F:A1:8C:54</fingerprint>
  <map-type>x509c2n:specified</map-type>
  <name>scooby-doo</name>
</cert-to-name>
</cert-maps>
</client-auth>
</tls>
</endpoint>
</listen>

<!-- calling home to a RESTCONF client -->
<call-home>
  <restconf-client>
    <name>config-manager</name>
  </restconf-client>
  <endpoints>
    <endpoint>
      <name>east-data-center</name>
      <address>22.33.44.55</address>
    </endpoint>
    <endpoint>
      <name>west-data-center</name>
      <address>33.44.55.66</address>
    </endpoint>
  </endpoints>
  <certificates>
    <certificate>ex-key-sect571r1-cert</certificate>
  </certificates>
  <client-auth>
    <trusted-ca-certs>
      deployment-specific-ca-certs
    </trusted-ca-certs>
    <trusted-client-certs>
      explicitly-trusted-client-certs
    </trusted-client-certs>
    <cert-maps>
      <cert-to-name>
        <id>1</id>
        <fingerprint>11:0A:05:11:00</fingerprint>
        <map-type>x509c2n:san-any</map-type>
      </cert-to-name>
      <cert-to-name>
        <fingerprint>11:0A:05:11:00</fingerprint>
        <map-type>x509c2n:san-any</map-type>
      </cert-to-name>
    </cert-maps>
  </client-auth>
</call-home>
<id>2</id>
<fingerprint>B3:4F:A1:8C:54</fingerprint>
<map-type>x509c2n:specified</map-type>
<name>scooby-doo</name>

</cert-to-name>
</cert-maps>
</client-auth>
</tls>
<connection-type>
<periodic>
<idle-timeout>300</idle-timeout>
<reconnect-timeout>60</reconnect-timeout>
</periodic>
</connection-type>
<reconnect-strategy>
<start-with>last-connected</start-with>
<max-attempts>3</max-attempts>
</reconnect-strategy>
</restconf-client>
</call-home>

</restconf-server>

4.5.3. YANG Model

This YANG module imports YANG types from [RFC6991] and [RFC7407].

<CODE BEGINS> file "ietf-restconf-server@2015-10-09.yang"

module ietf-restconf-server {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-restconf-server";
  prefix "rcserver";

  //import ietf-netconf-acm {
  //  prefix nacm;                     // RFC 6536
  //}
  import ietf-inet-types {           // RFC 6991
    prefix inet;
  }
  import ietf-x509-cert-to-name {    // RFC 7407
    prefix x509c2n;
  }
  import ietf-tls-server {           // RFC VVVV
    prefix ts;
    revision-date 2015-10-09;
  }

Watsen & Schoenwaelder Expires April 11, 2016 [Page 51]
This module contains a collection of YANG definitions for configuring RESTCONF servers.

Copyright (c) 2014 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 VVVV; see the RFC itself for full legal notices.

revision "2015-10-09" {
  description
    "Initial version";
  reference
    "RFC VVVV: NETCONF Server and RESTCONF Server Configuration Models";
}

// Features

feature tls-listen {


description
"The listen feature indicates that the RESTCONF server supports opening a port to listen for incoming RESTCONF client connections."
reference
"RFC XXXX: RESTCONF Protocol";
}

feature tls-call-home {
  description
  "The call-home feature indicates that the RESTCONF server supports initiating connections to RESTCONF clients."
  reference
  "RFC YYYY: NETCONF Call Home and RESTCONF Call Home"
}

feature client-cert-auth {
  description
  "The client-cert-auth feature indicates that the RESTCONF server supports the ClientCertificate authentication scheme."
  reference
  "RFC ZZZZ: Client Authentication over New TLS Connection"
}

// top-level container
container restconf-server {
  description
  "Top-level container for RESTCONF server configuration."

  container listen {
    if-feature tls-listen;
    description
      "Configures listen behavior";
    leaf max-sessions {
      type uint16;
      default 0;  // should this be 'max'?  
      description
        "Specifies the maximum number of concurrent sessions that can be active at one time. The value 0 indicates that no artificial session limit should be used."
    }
  }

  list endpoint {
    key name;
    description
      "List of endpoints to listen for RESTCONF connections on."
    leaf name {
      type string;
    }
  }
}
description
  "An arbitrary name for the RESTCONF listen endpoint."
}
choice transport {
  mandatory true;
  description
  "Selects between available transports.";
  case tls {
    if-feature tls-listen;
    container tls {
      description
      "TLS-specific listening configuration for inbound
      connections.";
      uses tls:listening-tls-server-grouping {
        refine port {
          default 443;
        }
        augment "client-auth" {
          description
          "Augments in the cert-to-name structure.";
          uses cert-maps-grouping;
        }
      }
    }
  }
}
}

container call-home {
  if-feature tls-call-home;
  description
  "Configures call-home behavior";
  list restconf-client {
    key name;
    description
    "List of RESTCONF clients the RESTCONF server is to
     initiate call-home connections to.";
    leaf name {
      type string;
      description
      "An arbitrary name for the remote RESTCONF client.";
    }
    choice transport {
      mandatory true;
      description
      "Selects between TLS and any transports augmented in.";
      case tls {

if-feature tls-call-home;
container tls {
  description
  "Specifies TLS-specific call-home transport configuration.";
  uses endpoints-container {
    refine endpoints/endpoint/port {
      default 9999;
    }
  }
  uses ts:non-listening-tls-server-grouping {
    augment "client-auth" {
      description
      "Augments in the cert-to-name structure.";
      uses cert-maps-grouping;
    }
  }
}
}

container connection-type {
  description
  "Indicates the RESTCONF client’s preference for how the
  RESTCONF server’s connection is maintained.";
  choice connection-type {
    description
    "Selects between available connection types.";
    case persistent-connection {
      container persistent {
        presence true;
        description
        "Maintain a persistent connection to the RESTCONF
        client. If the connection goes down, immediately
        start trying to reconnect to it, using the
        reconnection strategy.

        This connection type minimizes any RESTCONF client
        to RESTCONF server data-transfer delay, albeit at
        the expense of holding resources longer.";
      }
      container keep-alives {
        description
        "Configures the keep-alive policy, to proactively
        test the aliveness of the TLS client. An
        unresponsive TLS client will be dropped after
        approximately (max-attempts * max-wait) seconds.";
        reference
        "RFC YYYY: NETCONF Call Home and RESTCONF Call Home,"}
leaf max-wait {
    type uint16 {
        range "1..max";
    }
    units seconds;
    default 30;
    description
        "Sets the amount of time in seconds after which
        if no data has been received from the TLS
        client, a TLS-level message will be sent to
        test the aliveness of the TLS client.";
}
leaf max-attempts {
    type uint8;
    default 3;
    description
        "Sets the number of sequential keep-alive messages
        that can fail to obtain a response from the TLS
        client before assuming the TLS client is no
        longer alive.";
}
}
}
case periodic-connection {
    container periodic {
        presence true;
        description
            "Periodically connect to the RESTCONF client, so that
            the RESTCONF client may deliver messages pending for
            the RESTCONF server. The RESTCONF client is expected
to close the connection when it is ready to release
it, thus starting the RESTCONF server’s timer until
next connection.";
    leaf reconnect-timeout {
        type uint16 {
            range "1..max";
        }
        units minutes;
        default 60;
        description
            "The maximum amount of unconnected time the RESTCONF
server will wait before re-establishing a connection
to the RESTCONF client. The RESTCONF server may
initiate a connection before this time if desired
(e.g., to deliver a notification).";
    }
container reconnect-strategy {
  description "The reconnection strategy guides how a RESTCONF server reconnects to an RESTCONF client, after losing a connection to it, even if due to a reboot. The RESTCONF server starts with the specified endpoint and tries to connect to it max-attempts times before trying the next endpoint in the list (round robin).";
  leaf start-with {
    type enumeration {
      enum first-listed {
        description "Indicates that reconnections should start with the first endpoint listed.";
      }
      enum last-connected {
        description "Indicates that reconnections should start with the endpoint last connected to. If no previous connection has ever been established, then the first endpoint configured is used. RESTCONF servers SHOULD be able to remember the last endpoint connected to across reboots.";
      }
    }
    default first-listed;
    description "Specifies which of the RESTCONF client’s endpoints the RESTCONF server should start with when trying to connect to the RESTCONF client.";
  }
  leaf max-attempts {
    type uint8 {
      range "1..max";
    }
    default 3;
    description "Specifies the number times the RESTCONF server tries to connect to a specific endpoint before moving on to the next endpoint in the list (round robin).";
  }
}
grouping cert-maps-grouping {
  description
  "A grouping that defines a container around the
cert-to-name structure defined in RFC 7407.";
  container cert-maps {
    uses x509c2n:cert-to-name;
    description
    "The cert-maps container is used by a TLS-based RESTCONF
server to map the RESTCONF client’s presented X.509
certificate to a RESTCONF username. If no matching and
valid cert-to-name list entry can be found, then the
RESTCONF server MUST close the connection, and MUST NOT
accept RESTCONF messages over it.";
    reference
    "RFC XXXX: The RESTCONF Protocol";
  }
}

description
  "This grouping is used by tls container for call-home
configurations.";
container endpoints {
  description
  "Container for the list of endpoints.";
  list endpoint {
    key name;
    min-elements 1;
    ordered-by user;
    description
    "User-ordered list of endpoints for this RESTCONF client.
    Defining more than one enables high-availability.";
    leaf name {
      type string;
      description
      "An arbitrary name for this endpoint.";
    }
    leaf address {
      type inet:host;
      mandatory true;
      description
      "The IP address or hostname of the endpoint. If a
      hostname is configured and the DNS resolution results
      in more than one IP address, the RESTCONF server
will process the IP addresses as if they had been explicitly configured in place of the hostname.

leaf port {
  type inet:port-number;
  description
  "The IP port for this endpoint. The RESTCONF server will use the IANA-assigned well-known port if no value is specified."
}

</CODE ENDS>

5. Security Considerations

This section needs to be filled in...

6. IANA Considerations

This document registers two URIs in the IETF XML registry [RFC2119]. Following the format in [RFC3688], the following registrations are requested:

Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

This document registers two YANG modules in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the following registrations are requested: 
7. Other Considerations

The YANG modules define herein do not themselves support virtual routing and forwarding (VRF). It is expected that external modules will augment in VRF designations when needed.

8. Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): Andy Bierman, Martin Bjorklund, Benoit Claise, Mehmet Ersue, David Lamparter, Alan Luchuk, Ladislav Lhotka, Radek Krejci, Tom Petch, Phil Shafer, and Bert Wijnen.

Juergen Schoenwaelder and was partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

9. References
9.1. Normative References


9.2. Informative References

Appendix A. Change Log

A.1. 00 to 01
  o Restructured document so it flows better
  o Added trusted-ca-certs and trusted-client-certs objects into the
    ietf-system-tls-auth module

A.2. 01 to 02
  o removed the "one-to-many" construct
  o removed "address" as a key field
  o removed "network-manager" terminology
  o moved open issues to github issues
  o brought TLS client auth back into model

A.3. 02 to 03
  o fixed tree diagrams and surrounding text

A.4. 03 to 04
  o reduced the number of grouping statements
  o removed psk-maps and associated feature statements
  o added ability for listen/call-home instances to specify which
    host-keys/certificates (of all listed) to use
  o clarified that last-connected should span reboots
  o added missing "objectives" for selecting which keys to use,
    authenticating client-certificates, and mapping authenticated
    client-certificates to usernames
  o clarified indirect client certificate authentication
  o added keep-alive configuration for listen connections
  o added global-level NETCONF session parameters
A.5. 04 to 05

- Removed all refs to the old ietf-system-tls-auth module
- Removed YANG 1.1 style if-feature statements (loss some expressiveness)
- Removed the read-only (config false) lists of SSH host-keys and TLS certs
- Added an if-feature around session-options container
- Added ability to configure trust-anchors for SSH X.509 client certs
- Now imports by revision, per best practice
- Added support for RESTCONF server
- Added RFC Editor instructions

A.6. 05 to 06

- Removed feature statement on the session-options container (issue #21).
- Added NACM statements to YANG modules for sensitive nodes (issue #24).
- Fixed default RESTCONF server port value to be 443 (issue #26).
- Added client-cert-auth subtree to ietf-restconf-server module (issue #27).
- Updated draft-ietf-netmod-snmp-cfg reference to RFC 7407 (issue #28).
- Added description statements for groupings (issue #29).
- Added description for braces to tree diagram section (issue #30).
- Renamed feature from "rfc6187" to "ssh-x509-certs" (issue #31).

A.7. 06 to 07

- Replaced "application" with "NETCONF/RESTCONF client" (issue #32).
- Reverted back to YANG 1.1 if-feature statements (issue #34).
- Removed import by revisions (issue #36).
- Removed groupings only used once (issue #37).
- Removed upper-bound on hello-timeout, idle-timeout, and max-sessions (issue #38).
- Clarified that when no listen address is configured, the NETCONF/RESTCONF server will listen on all addresses (issue #41).
- Update keep-alive reference to new section in Call Home draft (issue #42).
- Modified connection-type/persistent/keep-alives/interval-secs default value, removed the connection-type/periodic/linger-secs node, and also removed the reconnect-strategy/interval-secs node (issue #43).
- Clarified how last-connected reconnection type should work across reboots (issue #44).
- Clarified how DNS-expanded hostnames should be processed (issue #45).
- Removed text on how to implement keep-alives (now in the call-home draft) and removed the keep-alive configuration for listen connections (issue #46).
- Clarified text for .../periodic-connection/timeout-mins (issue #47).
- Fixed description on the "trusted-ca-certs" leaf-list (issue #48).
- Added optional keychain-based solution in appendix A (issue #49).
- Fixed description text for the interval-secs leaf (issue #50).
- moved idle-time into the listen, persistent, and periodic subtrees (issue #51).
- put presence statements on containers where it makes sense (issue #53).

A.8. 07 to 08

- Per WG consensus, replaced body with the keychain-based approach described in -07’s Appendix.
Added a lot of introductory text, improved examples, and what not.

Appendix B. Open Issues

Please see: https://github.com/netconf-wg/server-model/issues.

Authors' Addresses

Kent Watsen
Juniper Networks
EMail: kwatsen@juniper.net

Juergen Schoenwaelder
Jacobs University Bremen
EMail: j.schoenwaelder@jacobs-university.de
Abstract

This document describes a YANG library, which provides information about all the YANG modules used by a device to represent management and protocol information. A YANG library can be shared by multiple protocols within the same device. Simple caching mechanisms are needed to allow clients to minimize retrieval of this information.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 20, 2016.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

1. Introduction

There is a need for standard mechanisms to identify the YANG modules and submodules that are in use by a server that utilizes YANG-based data abstractions. If a large number of YANG modules are utilized by the server, then the YANG library information needed can be relatively large. This information changes very infrequently, so it is important that clients be able to cache the YANG library and easily identify if their cache is out-of-date.

YANG library information can be different on every server, and can change at run-time or across a server reboot. Typically, a firmware upgrade is required to change the set of YANG modules used by a server.

The following information is needed by a client application (for each YANG module in the library) to fully utilize the YANG data modeling language:
o name: The mandatory YANG module name MUST be unique within a YANG library.

o revision: Each YANG module and submodule within the library has a revision. This is derived from the most recent revision statement within the module or submodule. If no such revision statement exists, the module’s or submodule’s revision is the empty string.

o submodule list: The name and revision of each submodule used by the module MUST be identified.

o feature list: The name of each YANG feature supported by the server MUST be identified.

o deviation list: The name of each YANG module used for deviation statements SHOULD be identified.

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

1.1.1. NETCONF

The following terms are defined in [RFC6241]:

o client

o server

1.1.2. YANG

The following terms are defined in [RFC6020]:

o module

o submodule

1.1.3. Terms

The following terms are used within this document:

o YANG library: a collection of YANG modules and submodules used by a server
1.1.4. Tree Diagrams

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

- Brackets "[" and "]" enclose list keys.
- Abbreviations before data node names: "rw" means configuration data (read-write) and "ro" state data (read-only).
- 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.

2. YANG Module Library

The "ietf-yang-library" module provides information about the YANG library used by a server.

YANG Tree Diagram for "ietf-yang-library" module:

```
+--ro modules
   +--ro module-set-id    string
   +--ro module* [name revision]
      +--ro name         yang:yang-identifier
      +--ro revision     union
      +--ro schema?      inet:uri
      +--ro namespace    inet:uri
      +--ro feature*     yang:yang-identifier
      +--ro deviation* [name revision]
         +--ro name        yang:yang-identifier
         +--ro revision     union
      +--ro conformance   enumeration
   +--ro submodules
      +--ro submodule* [name revision]
         +--ro name         yang:yang-identifier
         +--ro revision     union
         +--ro schema?      inet:uri

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

2.1.1. modules/module-set-id

This mandatory leaf contains a unique implementation-specific identifier representing the current set of modules and submodules. This can for example be a checksum of all modules and submodules.

This leaf allows a client to fetch the module list once, cache them, and only re-fetch them if the value of this leaf has been changed.

2.1.2. modules/module

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

2.2. YANG Library Module

The "ietf-yang-library" module defines monitoring information for the YANG modules used by a server.

The "ietf-yang-types" and "ietf-inet-types" modules from [RFC6991] are used by this module for some type definitions.

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

<CODE BEGINS> file "ietf-yang-library@2015-10-18.yang"
module ietf-yang-library {
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-library";
  prefix "yanglib";

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

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

  contact "WG Web:  <http://tools.ietf.org/wg/netconf/>
          WG List: <mailto:netconf@ietf.org>"
This module contains monitoring information about the YANG modules and submodules that are used within a YANG-based server.

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

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

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

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

// RFC Ed.: remove this note
// Note: extracted from draft-ietf-netconf-yang-library-02.txt

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
revision 2015-10-18 {
    description
        "Initial revision.";
    reference
        "RFC XXXX: YANG Module Library.";
}
typedef revision-identifier {
    type string {
        pattern '\d{4}-\d{2}-\d{2}';
    }
    description
        "Represents a specific date in YYYY-MM-DD format.";
}

grouping module {
    description
        "The module data structure is represented as a grouping
        so it can be reused in configuration or another monitoring
data structure.";
}

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

    leaf name {
        type yang:yang-identifier;
        description
            "The YANG module or submodule name.";
    }

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

grouping schema-leaf {
    description
        "Common schema leaf parameter for modules and submodules.";

    leaf schema {
        type inet:uri;
        description
            "Contains a URL that represents the YANG schema
            resource for this module or submodule.

            This leaf will only be present if there is a URL
            available for retrieval of the schema for this entry.";
    }
}
list module {
    key "name revision";
    description
        "Each entry represents one module currently
         supported by the server.";
    uses common-leafs;
    uses schema-leaf;
    leaf namespace {
        type inet:uri;
        mandatory true;
        description
            "The XML namespace identifier for this module.";
    }
    leaf-list feature {
        type yang:yang-identifier;
        description
            "List of YANG feature names from this module that are
             supported by the server.";
    }
    list deviation {
        key "name revision";
        description
            "List of YANG deviation module names and revisions
             used by this server to modify the conformance of
             the module associated with this entry. Note that
             the same module can be used for deviations for
             multiple modules, so the same entry MAY appear
             within multiple 'module' entries.
             If the deviation module is available for download
             from the server then a 'module' entry for that module
             will exist, with the same name and revision values.
             The 'conformance' value will be 'implement' for
             the deviation module.";
        uses common-leafs;
    }
    leaf conformance {
        type enumeration {
            enum implement {
                description
                    "Indicates the server implements one or more
                     protocol-accessible objects defined in the
                     YANG module identified in this entry. This includes
                     deviation statements defined in the module.";
            }
        }
    }
}
For YANG 1.1 modules, there MUST NOT be more than one module entry for a particular module name.

For YANG 1.1 modules that use the import statement without specifying a revision date, the implemented revision of the imported module MUST be used. If the imported module is not implemented, then the most recent revision of the imported module used by the server (and contained in the module list) MUST be used.

For YANG 1.0 modules, there SHOULD NOT be more than one module entry for a particular module name.

```yang
eum import {
  description
  "Indicates the server imports reusable definitions from the specified revision of the module, but does not implement any protocol accessible objects from this revision.

  Multiple module entries for the same module name MAY exist. This can occur if multiple modules import the same module, but specify different revision-dates in the import statements.

  For import statements that do not specify a revision date, the most recent revision in the library SHOULD be used by the server.";
}

container submodules {
  description
  "Contains information about all the submodules used by the parent module entry";

  list submodule {
    key "name revision";
    description
    "Each entry represents one submodule within the parent module.";
    uses common-leafs;
    uses schema-leaf;
  }
}

```
container modules {
    config false;
    description
        "Contains YANG module monitoring information."

    leaf module-set-id {
        type string;
        mandatory true;
        description
            "Contains a server-specific identifier representing
             the current set of modules and submodules. The
             server MUST change the value of this leaf if the
             information represented by the 'module' list instances
             has changed."
    }
    uses module;
}

<CODE ENDS>

3. IANA Considerations

3.1. YANG Module Registry

This document registers one URI in the IETF XML registry [RFC3688].
Following the format in RFC 3688, the following registration is
requested to be made.

Registrant Contact: The NETMOD WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

This document registers one YANG module in the YANG Module Names
registry [RFC6020].
4. Security Considerations

The 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 SSH [RFC6242].

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

- /modules/module: The module list used in a server implementation may help an attacker identify the server capabilities and server implementations with known bugs. Server vulnerabilities may be specific to particular modules, module revisions, module features, or even module deviations. This information is included in each module entry. For example, if a particular operation on a particular data node is known to cause a server to crash or significantly degrade device performance, then the module list information will help an attacker identify server implementations with such a defect, in order to launch a denial of service attack on the device.

5. Acknowledgements

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

6. Normative References


Appendix A. Change Log

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

A.1. 01 to 02
  o clarify ‘implement’ conformance for YANG 1.1 modules

A.2. 00 to 01
  o change conformance leaf to enumeration
  o filled in security considerations section

A.3. draft-ietf-netconf-restconf-03 to 00
  o moved ietf-yang-library from RESTCONF draft to new draft

Appendix B. Open Issues

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

The YANG Library issue tracker can be found here:

https://github.com/netconf-wg/yang-library/issues

Authors’ Addresses

Andy Bierman
YumaWorks

Email: andy@yumaworks.com
Abstract

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

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 20, 2016.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
# Table of Contents

1. Introduction ................................................. 3
   1.1. Terminology ........................................... 3
       1.1.1. NETCONF ........................................... 3
       1.1.2. HTTP ............................................... 4
       1.1.3. YANG ................................................ 4
       1.1.4. RESTCONF ........................................... 5
       1.1.5. YANG Patch .......................................... 5
       1.1.6. Tree Diagrams ....................................... 5
2. YANG Patch .................................................. 6
   2.1. Target Resource .......................................... 6
   2.2. yang-patch Input ......................................... 6
   2.3. yang-patch-status Output ................................. 7
   2.4. Target Data Node ......................................... 7
   2.5. Edit Operations ......................................... 8
   2.6. Error Handling .......................................... 8
   2.7. yang-patch RESTCONF Capability ........................... 9
3. YANG Module .................................................. 9
4. IANA Considerations .......................................... 18
   4.1. YANG Module Registry ..................................... 18
   4.2. application/yang.patch Media Types ....................... 18
   4.3. application/yang.patch-status Media Types ................ 19
   4.4. RESTCONF Capability URNs ................................ 19
5. Security Considerations ....................................... 19
6. Normative References ......................................... 20
Appendix A. Acknowledgements ................................... 21
Appendix B. Change Log .......................................... 21
   B.1. v05 to v06 ............................................. 21
   B.2. v04 to v05 ............................................. 21
   B.3. v03 to v04 ............................................. 21
   B.4. v02 to v03 ............................................. 22
   B.5. v01 to v02 ............................................. 22
   B.6. v00 to v01 ............................................. 22
   B.7. bierman:yang-patch-00 to ietf:yang-patch-00 .............. 23
Appendix C. Open Issues ......................................... 23
Appendix D. Example YANG Module ................................ 23
   D.1. YANG Patch Examples ..................................... 24
       D.1.1. Add Resources: Error ................................ 24
       D.1.2. Add Resources: Success ................................ 27
       D.1.3. Move list entry example ............................. 28
Authors’ Addresses ............................................. 29
1. Introduction

There is a need for standard mechanisms to patch NETCONF [RFC6241] datastores which contain conceptual data that conforms to schema specified with YANG [RFC6020]. An "ordered edit list" approach is needed to provide client developers with a simpler edit request format that can be more efficient and also allow more precise client control of the transaction procedure than existing mechanisms.

This document defines a media type for a YANG-based editing mechanism that can be used with the HTTP PATCH method [RFC5789]. YANG Patch is designed to support the RESTCONF protocol, defined in [I-D.ietf-netconf-restconf].

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

1.1.1. NETCONF

The following terms are defined in [RFC6241]:

- client
- configuration data
- datastore
- configuration datastore
- protocol operation
- running configuration datastore
- server
- state data
- user
1.1.2. HTTP

The following terms are defined in [RFC2616]:

- entity tag
- fragment
- header line
- message body
- method
- path
- query
- request URI
- response body

1.1.3. YANG

The following terms are defined in [RFC6020]:

- container
- data node
- key leaf
- leaf
- leaf-list
- list
- presence container (or P-container)
- RPC operation (now called protocol operation)
- non-presence container (or NP-container)
- ordered-by system
- ordered-by user
1.1.4.  RESTCONF

The following terms are defined in [I-D.ietf-netconf-restconf]:

- data resource
- datastore resource
- patch
- RESTCONF capability
- target resource

1.1.5.  YANG Patch

The following terms are used within this document:

- YANG Patch: a conceptual edit request using the "yang-patch" YANG container, defined in Section 3. In HTTP, refers to a PATCH method where the media type is "application/yang.patch+xml" or "application/yang.patch+json".

- YANG Patch Status: a conceptual edit status response using the YANG "yang-patch-status" container, defined in Section 3. In HTTP, refers to a response message for a PATCH method, where the message body is identified by the media type "application/yang.patch-status+xml" or "application/yang.patch-status+json".

1.1.6.  Tree Diagrams

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

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

A "YANG Patch" is an ordered list of edits that are applied to the target datastore by the server. The specific fields are defined in the YANG module in Section 3.

For RESTCONF, the YANG Patch operation is invoked by the client by sending a PATCH method request with the YANG Patch media type. A message body representing the YANG Patch input parameters MUST be provided.

The RESTCONF server MUST return the Accept-Patch header in an OPTIONS response, as specified in [RFC5789], which includes the media type for YANG Patch.

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

Example:

Accept-Patch: application/yang.patch

2.1. Target Resource

The YANG Patch operation uses a conceptual root within a NETCONF configuration datastore to identity the patch point for the edit operation. This root can be the datastore itself, or 1 or more data nodes within the datastore.

For RESTCONF, the target resource is derived from the request URI.

2.2. yang-patch Input

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

YANG Tree Diagram For "yang-patch" Container:

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

2.3. yang-patch-status Output

A data element representing the YANG Patch Status is returned to the client to report the detailed status of the edit operation. When used with the HTTP PATCH method, this data is identified by the YANG Patch Status media type, and the syntax specification is defined in Section 3.

YANG Tree Diagram For "yang-patch-status" Container:

```
+---rw yang-patch-status
  +---rw patch-id?   string
  +---rw (global-status)?
    |  +--:(global-errors)
    |     +---ro errors
    |  +--:(ok)
    |     +---rw ok?              empty
  +---rw edit-status
    +---rw edit [edit-id]
      +---rw edit-id     string
      +---rw (edit-status-choice)?
        |  +--:(ok)
        |     +---rw ok?              empty
        |  +--:(errors)
        |     +---ro errors
```

2.4. Target Data Node
The target data node for each edit operation is determined by the value of the target resource in the request and the "target" leaf within each "edit" entry.

If the target resource specified in the request URI identifies a datastore resource, then the path string in the "target" leaf is an absolute path expression. The first node specified in the "target" leaf is a top-level data node defined within a YANG module.

If the target resource specified in the request URI identifies a data resource, then the path string in the "target" leaf is a relative path expression. The first node specified in the "target" leaf is a child node of the data node associated with the target resource.

2.5. Edit Operations

Each YANG patch edit specifies one edit operation on the target data node. The set of operations is aligned with the NETCONF edit operations, but also includes some new operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>create a new data resource if it does not already exist or error</td>
</tr>
<tr>
<td>delete</td>
<td>delete a data resource if it already exists or error</td>
</tr>
<tr>
<td>insert</td>
<td>insert a new user-ordered data resource</td>
</tr>
<tr>
<td>merge</td>
<td>merge the edit value with the target data resource; create if it does not already exist</td>
</tr>
<tr>
<td>move</td>
<td>re-order the target data resource</td>
</tr>
<tr>
<td>replace</td>
<td>replace the target data resource with the edit value</td>
</tr>
<tr>
<td>remove</td>
<td>remove a data resource if it already exists or no error</td>
</tr>
</tbody>
</table>

YANG Patch Edit Operations

2.6. Error Handling

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

All the specified edits MUST be applied or the target datastore contents MUST be returned to its original state before the PATCH method started.
The server will save the running datastore to non-volatile storage if it has changed, after the edits have been successfully completed.

2.7. yang-patch RESTCONF Capability

A URI is defined to identify the YANG Patch extension to the base RESTCONF protocol. If the server supports the YANG Patch media type, then the "yang-patch" RESTCONF capability defined in Section 4.4 MUST be present in the "capability" leaf-list in the "ietf-restconf-monitoring" module defined in [I-D.ietf-netconf-restconf].

3. YANG Module

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

The "ietf-restconf" module from [I-D.ietf-netconf-restconf] is used by this module for the 'restconf-media-type' extension definition.

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

<CODE BEGINS> file "ietf-yang-patch@2015-04-30.yang"

module ietf-yang-patch {
    prefix "ypatch";

    import ietf-yang-types { prefix yang; }

    import ietf-restconf {
        prefix rc;
        revision-date 2015-10-07;
    }

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

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

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

        WG Chair: Mahesh Jethanandani

This module contains conceptual YANG specifications for the YANG Patch and YANG Patch Status data structures.

Note that the YANG definitions within this module do not represent configuration data of any kind. The YANG grouping statements provide a normative syntax for XML and JSON message encoding purposes.

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

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

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

revision 2015-06-04 {
  description "Initial revision."
  reference "RFC XXXX: YANG Patch Media Type."
}
typedef target-resource-offset {
  type yang:xpath1.0;
  description
    "Contains an XPath absolute path expression identifying
    a sub-resource within the target resource.
    The document root for this XPath expression is the
    target resource that is specified in the
    protocol operation (e.g., the URI for the PATCH request).";
}

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

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

grouping yang-patch {
  description
    "A grouping that contains a YANG container
    representing the syntax and semantics of a
    YANG Patch edit request message.";

container yang-patch {
  description
    "Represents a conceptual sequence of datastore edits,
    called a patch. Each patch is given a client-assigned
    patch identifier. Each edit MUST be applied
    in ascending order, and all edits MUST be applied.
    If any errors occur, then the target datastore MUST NOT
    be changed by the patch operation.

    YANG datastore validation (defined in RFC 6020, section
    8.3.3) is performed before any edits have been applied
    to the running datastore.

    It is possible for a datastore constraint violation to occur
    due to any node in the datastore, including nodes not
    included in the edit list. Any validation errors MUST
    be reported in the reply message.";
eference
  "RFC 6020, section 8.3.";

  leaf patch-id {
    type string;
    
"An arbitrary string provided by the client to identify the entire patch. This value SHOULD be present in any audit logging records generated by the server for the patch. Error messages returned by the server pertaining to this patch will be identified by this patch-id value."
}

leaf comment {
  type string;
  description
  "An arbitrary string provided by the client to describe the entire patch. This value SHOULD be present in any audit logging records generated by the server for the patch."
}

list edit {
  key edit-id;
  ordered-by user;
  description
  "Represents one edit within the YANG Patch request message. The edit list is applied in the following manner:
  - The first edit is conceptually applied to a copy of the existing target datastore, e.g., the running configuration datastore.
  - Each ascending edit is conceptually applied to the result of the previous edit(s).
  - After all edits have been successfully processed, the result is validated according to YANG constraints.
  - If successful, the server will attempt to apply the result to the target datastore. ";

  leaf edit-id {
    type string;
    description
    "Arbitrary string index for the edit. Error messages returned by the server pertaining to a specific edit will be identified by this value."
  }

  leaf operation {
    type enumeration {
      enum create {

description
"The target data node is created using the supplied value, only if it does not already exist."
}
enum delete {
  description
  "Delete the target node, only if the data resource currently exists, otherwise return an error."
}
enum insert {
  description
  "Insert the supplied value into a user-ordered list or leaf-list entry. The target node must represent a new data resource. If the 'where' parameter is set to 'before' or 'after', then the 'point' parameter identifies the insertion point for the target node."
}
enum merge {
  description
  "The supplied value is merged with the target data node."
}
enum move {
  description
  "Move the target node. Reorder a user-ordered list or leaf-list. The target node must represent an existing data resource. If the 'where' parameter is set to 'before' or 'after', then the 'point' parameter identifies the insertion point to move the target node."
}
enum replace {
  description
  "The supplied value is used to replace the target data node."
}
enum remove {
  description
  "Delete the target node if it currently exists."
}

mandatory true;
description
"The datastore operation requested for the associated edit entry";
leaf target {
    type target-resource-offset;
    mandatory true;
    description
        "Identifies the target data resource for the edit operation."
}

leaf point {
    when "(./operation = 'insert' or " + 
        "../operation = 'move') and " + 
        "(./where = 'before' or 
        "../where = 'after')" {
        description
            "Point leaf only applies for insert or move operations, before or after an existing entry."
    }
    type target-resource-offset;
    description
        "The absolute URL path for the data node that is being used as the insertion point or move point for the target of this edit entry."
}

leaf where {
    when "./operation = 'insert' or 
        "../operation = 'move'" {
        description
            "Where leaf only applies for insert or move operations."
    }
    type enumeration {
        enum before {
            description
                "Insert or move a data node before the data resource identified by the 'point' parameter."
        }
        enum after {
            description
                "Insert or move a data node after the data resource identified by the 'point' parameter."
        }
        enum first {
            description
                "Insert or move a data node so it becomes ordered as the first entry."
        }
        enum last {
            description
                "Insert or move a data node so it becomes ordered..."
as the last entry."
}
}
default last;
description
"Identifies where a data resource will be inserted or
moved. YANG only allows these operations for
list and leaf-list data nodes that are ordered-by
user."
}

anyxml value {
  when "(.operation = 'create' or " + 
  "../operation = 'merge' " + 
  "or ../operation = 'replace' or " + 
  "../operation = 'insert')" {
    description
    "Value node only used for create, merge,
    replace, and insert operations"
  }
}
description
"Value used for this edit operation.
The anyxml value MUST represent a container with
exactly one child node, which MUST identify the
target resource associated with the 'target' leaf.

The descendants of this node MUST NOT contain
an 'anyxml' data node. Only 'list', 'container',
'leaf', and 'leaf-list' data nodes can appear as
descendant nodes of this object.

For example, suppose the target node is a YANG container
named foo:

    container foo {
      leaf a { type string; }
      leaf b { type int32; }
    }

The value node will contain one instance of foo:

    <value>
      <foo xmlns='example-foo-namespace'>
        <a>some value</a>
        <b>42</b>
      </foo>
    </value>
grouping yang-patch-status {
  description
  "A grouping that contains a YANG container representing the syntax and semantics of YANG Patch status response message.";

  container yang-patch-status {
    description
    "A container representing the response message sent by the server after a YANG Patch edit request message has been processed.";

    leaf patch-id {
      type string;
      description
      "The patch-id value used in the request";
    }

    choice global-status {
      description
      "Report global errors or complete success. If there is no case selected then errors are reported in the edit-status container.";

      case global-errors {
        uses rc:errors;
        description
        "This container will be present if global errors unrelated to a specific edit occurred.";
      }
      leaf ok {
        type empty;
        description
        "This leaf will be present if the request succeeded and there are no errors reported in the edit-status container.";
      }
    }
  }
}

container edit-status {
    description
    "This container will be present if there are
    edit-specific status responses to report.
    If all edits succeeded and the 'global-status'
    returned is 'ok', then a server MAY omit this
    container";
}

description
"Represents a list of status responses,
corresponding to edits in the YANG Patch
request message. If an edit entry was
skipped or not reached by the server,
then this list will not contain a corresponding
entry for that edit.";

description
"Response status is for the edit list entry
with this edit-id value.";

description
"A choice between different types of status
responses for each edit entry.";

leaf ok {
    type empty;
    description
    "This edit entry was invoked without any
    errors detected by the server associated
    with this edit.";
}

case errors {
    uses rc:errors;
    description
    "The server detected errors associated with the
    edit identified by the same edit-id value.";
}

} // grouping yang-patch-status
4.  IANA Considerations

4.1.  YANG Module Registry

This document registers one URI in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made.

Registrant Contact: The NETMOD WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

This document registers one YANG module in the YANG Module Names registry [RFC6020].

name:        ietf-yang-patch
prefix:      ypatch
// RFC Ed.: replace XXXX with RFC number and remove this note
reference:   RFC XXXX

4.2.  application/yang.patch Media Types

The MIME media type for a YANG Patch document is application/yang.patch.

  Type name: application
  Subtype name: yang.patch
  Required parameters: none
  Optional parameters: none
  Encoding considerations: 8-bit
  Security considerations: See Section 5 of RFC XXXX
  Interoperability considerations: none

  // RFC Ed.: replace XXXX with RFC number and remove this note
  Published specification: RFC XXXX
4.3. application/yang.patch-status Media Types

The MIME media type for a YANG Patch status document is application/yang.patch-status.

   Type name: application
   Subtype name: yang.patch-status
   Required parameters: none
   Optional parameters: none
   Encoding considerations: 8-bit
   Security considerations: See Section 5 of RFC XXXX
   Interoperability considerations: none

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

4.4. RESTCONF Capability URNs

This document registers one capability identifier in "RESTCONF Protocol Capability URNs" registry

   Index
       Capability Identifier
           ------------------------

       :yang-patch
           urn:ietf:params:restconf:capability:yang-patch:1.0

5. Security Considerations

The YANG Patch media type does not introduce any significant new security threats, beyond what is described in
[I-D.ietf-netconf-restconf]. This document defines edit processing instructions for a variant of the PATCH method, as used within the
RESTCONF protocol.

It is important for server implementations to carefully validate all the edit request parameters in some manner. If the entire YANG Patch
request cannot be completed, then no configuration changes to the system are done.
A server implementation SHOULD attempt to prevent system disruption due to partial processing of the YANG Patch edit list. It may be possible to construct an attack on such a server, which relies on the edit processing order mandated by YANG Patch.

6. Normative References

[I-D.ietf-netconf-restconf]

[I-D.ietf-netmod-yang-json]
Lhotka, L., "JSON Encoding of Data Modeled with YANG", draft-ietf-netmod-yang-json-06 (work in progress), October 2015.

[I-D.ietf-netmod-yang-metadata]
Lhotka, L., "Defining and Using Metadata with YANG", draft-ietf-netmod-yang-metadata-02 (work in progress), September 2015.


[W3C.REC-xml-20081126]
Appendix A. Acknowledgements

The authors would like to thank the following people for their contributions to this document: Rex Fernando.

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

Appendix B. Change Log

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

The YANG Patch issue tracker can be found here: https://github.com/netconf-wg/yang-patch/issues

B.1. v05 to v06

- changed errors example so a full request and error response is shown in XML format
- fixed error-path to match instance-identifier encoding for both XML and JSON
- added references for YANG to JSON and YANG Metadata drafts
- clarified that YANG JSON drafts are used for encoding, not plain JSON

B.2. v04 to v05

- updated reference to RESTCONF

B.3. v03 to v04

- removed NETCONF specific text
- changed data-resource-offset typedef from a relative URI to an XPath absolute path expression
Internet-Draft                 YANG Patch                   October 2015

- clarified insert operation
- removed requirement that edits MUST be applied in ascending order
- change SHOULD keep datastore unchanged on error to MUST (this is required by HTTP PATCH)
- removed length restriction on ‘comment’ leaf
- updated YANG tree for example-jukebox library

B.4. v02 to v03

- added usage of restconf-media-type extension to map the yang-patch and yang-patch-status groupings to media types
- added yang-patch RESTCONF capability URI
- Added sub-section for terms used from RESTCONF
- filled in security considerations section

B.5. v01 to v02

- Reversed order of change log
- Clarified anyxml structure of "value" parameter within a YANG patch request (github issue #1)
- Updated RESTCONF reference
- Added note to open issues section to check github instead

B.6. v00 to v01

- Added text requiring support for Accept-Patch header, and removed ’Identification of YANG Patch capabilities’ open issue.
- Removed ’location’ leaf from yang-patch-status grouping
- Removed open issue ’Protocol independence’ because the location leaf was removed.
- Removed open issue ’RESTCONF coupling’ because there is no concern about a normative reference to RESTCONF. There may need to be a YANG 1.1 mechanism to allow protocol template usage (instead of grouping wrapper).
o Removed open issue ‘Is the delete operation needed’. It was decided that both delete and remove should remain as operations and clients can choose which one to use. This is not an implementation burden on the server.

o Removed open issue ‘global-errors needed’. It was decided that they are needed as defined because the global <ok/> is needed and the special key value for edit=global error only allows for 1 global error.

o Removed open issue ‘Is location leaf needed’. It was decided that it is not needed so this leaf has been removed.

o Removed open issue ‘Bulk editing support in yang-patch-status’. The ‘location’ leaf has been removed so this issue is no longer applicable.

o Removed open issue ‘Edit list mechanism’. Added text to the ‘edit’ list description-stmt about how the individual edits must be processed. There is no concern about duplicate edits which cause intermediate results to be altered by subsequent edits in the same edit list.

B.7. bierman:yang-patch-00 to ietf:yang-patch-00

o Created open issues section

Appendix C. Open Issues

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

Refer to the github issue tracker for any open issues:

https://github.com/netconf-wg/yang-patch/issues

Appendix D. Example YANG Module

The example YANG module used in this document represents a simple media jukebox interface. The "example-jukebox" YANG module is defined in [I-D.ietf-netconf-restconf].

YANG Tree Diagram for "example-jukebox" Module:

```
  +--rw jukebox!
    +--rw library
      |   +--rw artist* [name]
      |       |   +--rw name string
      |       |   +--rw album* [name]
```

D.1. YANG Patch Examples

This section includes RESTCONF examples. Most examples are shown in JSON encoding [RFC7158], and some are shown in XML encoding [W3C.REC-xml-20081126].

D.1.1. Add Resources: Error

The following example shows several songs being added to an existing album. Each edit contains one song. The first song already exists, so an error will be reported for that edit. The rest of the edits were not attempted, since the first edit failed. The XML encoding is used in this example.

Request from client:

```
PATCH /restconf/data/example-jukebox:jukebox/
library/artist=Foo%20Fighters/album=Wasting%20Light HTTP/1.1
```
<yang-patch xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-patch">
  <patch-id>add-songs-patch</patch-id>
  <edit>
    <edit-id>edit1</edit-id>
    <operation>create</operation>
    <target>/song</target>
    <value>
      <song xmlns="http://example.com/ns/example-jukebox">
        <name>Bridge Burning</name>
        <location>/media/bridge_burning.mp3</location>
        <format>MP3</format>
        <length>288</length>
      </song>
    </value>
  </edit>
  <edit>
    <edit-id>edit2</edit-id>
    <operation>create</operation>
    <target>/song</target>
    <value>
      <song xmlns="http://example.com/ns/example-jukebox">
        <name>Rope</name>
        <location>/media/rope.mp3</location>
        <format>MP3</format>
        <length>259</length>
      </song>
    </value>
  </edit>
  <edit>
    <edit-id>edit3</edit-id>
    <operation>create</operation>
    <target>/song</target>
    <value>
      <song xmlns="http://example.com/ns/example-jukebox">
        <name>Dear Rosemary</name>
        <location>/media/dear_rosemary.mp3</location>
        <format>MP3</format>
        <length>269</length>
      </song>
    </value>
  </edit>
</yang-patch>
XML Response from server:

HTTP/1.1 409 Conflict
Date: Mon, 23 Apr 2012 13:01:20 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
Content-Type: application/yang.patch-status+xml

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

JSON Response from server:

The following response is shown in JSON format to highlight the
difference in the "error-path" object encoding. For JSON, the
instance-identifier encoding in the "JSON Encoding of YANG
Data" draft is used. The "error-path" string is wrapped for
display purposes.

HTTP/1.1 409 Conflict
Date: Mon, 23 Apr 2012 13:01:20 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
Content-Type: application/yang.patch-status+json
D.1.2. Add Resources: Success

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

- Each of 2 edits contains one song.
- Both edits succeed and new sub-resources are created

Request from client:

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

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

"edit" : [

    {
        "edit-id": "edit1",
        "operation": "create",
        "target": "/song",
        "value": {
            "song": {
                "name": "Rope",
                "location": "/media/rope.mp3",
                "format": "MP3",
                "length": 259
            }
        }
    },

    {
        "edit-id": "edit2",
        "operation": "create",
        "target": "/song",
        "value": {
            "song": {
                "name": "Dear Rosemary",
                "location": "/media/dear_rosemary.mp3",
                "format": "MP3",
                "length": 269
            }
        }
    }
]

Response from server:

HTTP/1.1 200 Success
Date: Mon, 23 Apr 2012 13:01:20 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
Content-Type: application/yang.patch-status+json

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

D.1.3. Move list entry example
The following example shows a song being moved within an existing playlist. Song "1" in playlist "Foo-One" is being moved after song "3" in the playlist. The operation succeeds, so a non-error reply example can be shown.

Request from client:

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

{
    "ietf-yang-patch:yang-patch" : {
        "patch-id" : "move-song-patch",
        "comment" : "Move song 1 after song 3",
        "edit" : [
            {
                "edit-id" : "edit1",
                "operation" : "move",
                "target" : "/song/1",
                "point" : "/song3",
                "where" : "after"
            }
        ]
    }
}

Response from server:

HTTP/1.1 400 OK
Date: Mon, 23 Apr 2012 13:01:20 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
Content-Type: application/yang.patch-status+json

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

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
Abstract

This document defines a subscription and push mechanism for YANG datastores. This mechanism allows client applications to request updates from a YANG datastore, which are then pushed by the server to the client per a subscription policy, without requiring additional client requests.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 17, 2016.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction .................................................. 3
2. Definitions and Acronyms ..................................... 5
3. Solution Overview .............................................. 6
   3.1. Subscription Model ....................................... 7
   3.2. Negotiation of Subscription Policies .................... 9
   3.3. On-Change Considerations ................................. 9
   3.4. Data Encodings ........................................... 10
   3.4.1. Periodic Subscriptions ................................. 11
   3.4.2. On-Change Subscriptions ................................ 11
   3.5. Subscription Filters ..................................... 12
   3.6. Push Data Stream and Transport Mapping ................. 12
   3.7. Subscription management ................................... 16
   3.7.1. Subscription management by RPC ....................... 16
   3.7.2. Subscription management by configuration ............. 18
   3.8. Other considerations ..................................... 18
   3.8.1. Authorization ........................................... 18
   3.8.2. Additional subscription primitives .................... 19
   3.8.3. Robustness and reliability considerations ............ 19
   3.8.4. Update size and fragmentation considerations ......... 19
   3.8.5. Additional data streams ................................ 19
   3.8.6. Implementation considerations ......................... 20
4. A YANG data model for management of datastore push subscriptions .......................................... 21
   4.1. Overview .................................................. 21
   4.2. System streams ............................................. 23
   4.3. Filters .................................................... 23
   4.4. Subscription configuration ................................. 23
   4.5. Subscription monitoring .................................... 25
   4.6. Notifications ............................................. 25
   4.7. RPCs ..................................................... 26
   4.7.1. Create-subscription RPC ................................. 26
1. Introduction

YANG [RFC6020] was originally designed for the Netconf protocol [RFC6241], which originally put most emphasis on configuration. However, YANG is not restricted to configuration data. YANG datastores, i.e. datastores that contain data modeled according using YANG, can contain configuration as well as operational data. It is therefore reasonable to expect that data in YANG datastores will increasingly be used to support applications that are not focused on managing configurations but that are, for example, related to service assurance.

Service assurance applications typically involve monitoring operational state of networks and devices; of particular interest are changes that this data undergoes over time. Likewise, there are applications in which data and objects from one datastore need to be made available both to applications in other systems and to remote datastores [I-D.voit-netmod-peer-mount-requirements] [I-D.clemm-netmod-mount]. This requires mechanisms that allow remote systems to become quickly aware of any updates to allow to validate and maintain cross-network integrity and consistency.

Traditional approaches to remote network state visibility rely heavily on polling. With polling, data is periodically explicitly retrieved by a client from a server to stay up-to-date.

There are various issues associated with polling-based management:

- It introduces additional load on network, devices, and applications. Each polling cycle requires a separate yet arguably redundant request that results in an interrupt, requires parsing, consumes bandwidth.

- It lacks robustness. Polling cycles may be missed, requests may be delayed or get lost, often particularly in cases when the network is under stress and hence exactly when the need for the data is the greatest.
Data may be difficult to calibrate and compare. Polling requests may undergo slight fluctuations, resulting in intervals of different lengths which makes data hard to compare. Likewise, pollers may have difficulty issuing requests that reach all devices at the same time, resulting in offset polling intervals which again make data hard to compare.

A more effective alternative is when an application can request to be automatically updated as necessary of current content of the datastore (such as a subtree, or data in a subtree that meets a certain filter condition), and in which the server that maintains the datastore subsequently pushes those updates. However, such a solution does not currently exist.

The need to perform polling-based management is typically considered an important shortcoming of management applications that rely on MIBs polled using SNMP [RFC1157]. However, without a provision to support a push-based alternative, there is no reason to believe that management applications that operate on YANG datastores using protocols such as NETCONF or Restconf [I-D.ietf-netconf-restconf] will be any more effective, as they would follow the same request/response pattern.

While YANG allows the definition of notifications, such notifications are generally intended to indicate the occurrence of certain well-specified event conditions, such as a the onset of an alarm condition or the occurrence of an error. A capability to subscribe to and deliver event notifications has been defined in [RFC5277]. In addition, configuration change notifications have been defined in [RFC6470]. These change notifications pertain only to configuration information, not to operational state, and convey the root of the subtree to which changes were applied along with the edits, but not the modified data nodes and their values.

Accordingly, there is a need for a service that allows client applications to subscribe to updates of a YANG datastore and that allows the server to push those updates. The requirements for such a service are documented in [I-D.i2rs-pub-sub-requirements]. This document proposes a solution that features the following capabilities:

- A mechanism that allows clients to subscribe to automatic datastore updates, and the means to manage those subscriptions. The subscription allows clients to specify which data they are interested in, and to provide optional filters with criteria that data must meet for updates to be sent. Furthermore, subscription can specify a policy that directs when updates are provided. For
example, a client may request to be updated periodically in certain intervals, or whenever data changes occur.

- The ability for a server to push back on requested subscription parameters. Because not every server may support every requested interval for every piece of data, it is necessary for a server to be able to indicate whether or not it is capable of supporting a requested subscription, and possibly allow to negotiate subscription parameters.

- A mechanism to communicate the updates themselves. For this, the proposal leverages and extends existing YANG/Netconf/Restconf mechanisms, defining special notifications that carry updates.

This document specifies a YANG data model to manage subscriptions to data in YANG datastores, and to configure associated filters and data streams. It defines extensions to RPCs defined in [RFC5277] that allow to extend notification subscriptions to subscriptions for datastore updates. It also defines a notification that can be used to carry data updates and thus serve as push mechanism.

2. Definitions and Acronyms

Data node: An instance of management information in a YANG datastore.

Data record: A record containing a set of one or more data node instances and their associated values.

Datastore: A conceptual store of instantiated management information, with individual data items represented by data nodes which are arranged in hierarchical manner.

Datastream: A continuous stream of data records, each including a set of updates, i.e. data node instances and their associated values.

Data subtree: An instantiated data node and the data nodes that are hierarchically contained within it.

NACM: NETCONF Access Control Model

NETCONF: Network Configuration Protocol

Push-update stream: A conceptual data stream of a datastore that streams the entire datastore contents continuously and perpetually.

RPC: Remote Procedure Call

SNMP: Simple Network Management Protocol
Subscription: A contract between a client ("subscriber") and a server ("publisher"), stipulating which information the client wishes to receive from the server (and which information the server has to provide to the client) without the need for further solicitation.

Subscription filter: A filter that contains evaluation criteria which are evaluated against YANG objects of a subscription. An update is only published if the object meets the specified filter criteria.

Subscription policy: A policy that specifies under what circumstances to push an update, e.g. whether updates are to be provided periodically or only whenever changes occur.

Update: A data item containing the current value of a data node.

Update trigger: A trigger, as specified by a subscription policy, that causes an update to be sent, respectively a data record to be generated. An example of a trigger is a change trigger, invoked when the value of a data node changes or a data node is created or deleted, or a time trigger, invoked after the laps of a periodic time interval.

URI: Uniform Resource Identifier

YANG: A data definition language for NETCONF

Yang-push: The subscription and push mechanism for YANG datastores that is specified in this document.

3. Solution Overview

This document specifies a solution that allows clients to subscribe to information updates in a YANG datastore, which are subsequently pushed from the server to the client.

Subscriptions are initiated by clients. Servers respond to a subscription request explicitly positively or negatively. Negative responses include information about why the subscription was not accepted, in order to facilitate converging on an acceptable set of subscription parameters. Once a subscription has been established, datastore push updates are pushed from the server to the subscribing client until the subscription ends.

Accordingly, the solution encompasses several components:

- The subscription model for configuration and management of the subscriptions, with a set of associated services.
The ability to provide hints for acceptable subscription parameters, in cases where a subscription desired by a client cannot currently be served.

The stream of datastore push updates.

In addition, there are a number of additional considerations, such as the tie-in of the mechanisms with security mechanisms. Each of those aspects will be discussed in the following subsections.

3.1. Subscription Model

Yang-push subscriptions are defined using a data model that is itself defined in YANG. This model is based on the subscriptions defined in [RFC5277], which are also reused in Restconf. The model is extended with several parameters, including a subscription type and a subscription ID.

The subscription model assumes the presence of a conceptual perpetual datastream "push-update" of continuous datastore updates that can be subscribed to, although other datastreams may be supported as well. A subscription refers to a datastream and specifies filters that are to be applied to, it for example, to provide only those subsets of the information that match a filter criteria. In addition, a subscription specifies a set of subscription parameters that define the trigger when data records should be sent, for example at periodic intervals or whenever underlying data items change.

The complete set of subscription parameters is as follows:

- The stream being subscribed to. The subscription model assumes the presence of perpetual and continuous streams of updates. The stream "push-update" is always available and covers the entire set of YANG data in the server, but a system may provide other streams to choose from.

- The datastore to target. By default, the datastore will always be "running". However, it is conceivable that implementations want to also support subscriptions to updates to other datastores.

- An encoding for the data updates. By default, updates are encoded using XML, but JSON can be requested as an option and other encodings may be supported in the future.

- An optional start time for the subscription. If the specified start time is in the past, the subscription goes into effect immediately. The start time also serves as anchor time for
periodic subscriptions, from which intervals at which to send updates are calculated (see also below).

- An optional stop time for the subscription. Once the stop time is reached, the subscription is automatically terminated.

- A subscription policy definition regarding the update trigger when to send new updates. The trigger can be periodic or based on change.

  * For periodic subscriptions, the trigger is defined by a parameter that defines the interval with which updates are to be pushed. The start time of the subscription serves as anchor time, defining one specific point in time at which an update needs to be sent. Update intervals always fall on the points in time that are a multiple of a period after the start time.

  * For on-change subscriptions, the trigger occurs whenever a change in the subscribed information is detected. On-change subscriptions have more complex semantics that can be guided by additional parameters. Please refer also to Section 3.3.

    + One parameter is needed to specify the dampening period, i.e. the interval that must pass before a successive update for the same data node is sent. The first time a change is detected, the update is sent immediately. If a subsequent change is detected, another update is only sent once the dampening period has passed, containing the value of the data node that is then valid.

    + Another parameter allows to restrict the types of changes for which updates are sent (changes to object values, object creation or deletion events). It is conceivable to augment the data model with additional parameters in the future to specify even more refined policies, such as parameters that specify the magnitude of a change that must occur before an update is triggered.

    + A third parameter specifies whether or not a complete update with all the subscribed data should be sent at the beginning of a subscription.

- Optionally, a filter, or set of filters, describing the subset of data items in the stream's data records that are of interest to the subscriber. The server should only send to the subscriber the data items that match the filter(s), when present. The absence of a filter indicates that all data items from the stream are of interest to the subscriber and all data records must be sent in
their entirety to the subscriber. Two types of filters are supported: subtree filter, with the same semantics as defined in [RFC 6241], and XPath filters. Additional filter types can be added through augmentations. Filters can be specified "inline" as part of the subscription, or can be configured separately and referenced by a subscription, in order to facilitate reuse of complex filters.

The subscription data model is specified as part of the YANG data model described later in this specification. Specifically, the subscription parameters are defined in the "subscription-info" grouping. It is conceivable that additional subscription parameters might be added in the future. This can be accomplished through augmentation of the subscription data model.

3.2. Negotiation of Subscription Policies

A subscription rejection can be caused by the inability of the server to provide a stream with the requested semantics. For example, a server may not be able to support "on-change" updates for operational data, or only support them for a limited set of data nodes. Likewise, a server may not be able to support a requested updated frequency, or a requested encoding.

Yang-push supports a simple negotiation between clients and servers for subscription parameters. The negotiation is limited to a single pair of subscription request and response. For negative responses, the server SHOULD include in the returned error what subscription parameters would have been accepted for the request. The returned acceptable parameters constitute suggestions that, when followed, increase the likelihood of success for subsequent requests. However, they are no guarantee that subsequent requests for this client or others will in fact be accepted.

In case a subscriber requests an encoding other than XML, and this encoding is not supported by the server, the server simply indicates in the response that the encoding is not supported.

3.3. On-Change Considerations

On-change subscriptions allow clients to subscribe to updates whenever changes to objects occur. As such, on-change subscriptions are of particular interest for data that changes relatively infrequently, yet that require applications to be notified with minimal delay when changes do occur.

On-change subscriptions tend to be more difficult to implement than periodic subscriptions. Specifically, on-change subscriptions may
involve a notion of state to see if a change occurred between past and current state, or the ability to tap into changes as they occur in the underlying system. Accordingly, on-change subscriptions may not be supported by all implementations or for every object.

When an on-change subscription is requested for a datastream with a given subtree filter, where not all objects support on-change update triggers, the subscription request MUST be rejected. As a result, on-change subscription requests will tend to be directed at very specific, targeted subtrees with only few objects.

Any updates for an on-change subscription will include only objects for which a change was detected. To avoid flooding clients with repeated updates for fast-changing objects, or objects with oscillating values, an on-change subscription allows for the definition of a dampening period. Once an update for a given object is sent, no other updates for this particular object are sent until the end of the dampening period. In addition, updates include information about objects that were deleted and ones that were newly created.

On-change subscriptions can be refined to let users subscribe only to certain types of changes, for example, only to object creations and deletions, but not to modifications of object values.

Additional refinements are conceivable. For example, in order to avoid sending updates on objects whose values undergo only a negligible change, additional parameters might be added to an on-change subscription specifying a policy that states how large or "significant" a change has to be before an update is sent. A simple policy is a "delta-policy" that states, for integer-valued data nodes, the minimum difference between the current value and the value that was last reported that triggers an update. Also more sophisticated policies are conceivable, such as policies specified in percentage terms or policies that take into account the rate of change. While not specified as part of this draft, such policies can be accommodated by augmenting the subscription data model accordingly.

3.4. Data Encodings

Subscribed data is encoded in either XML or JSON format. A server MUST support XML encoding and MAY support JSON encoding.

It is conceivable that additional encodings may be supported as options in the future. This can be accomplished by augmenting the subscription data model with additional identity statements used to refer to requested encodings.
3.4.1. Periodic Subscriptions

In a periodic subscription, the data included as part of an update corresponds to data that could have been simply retrieved using a get operation and is encoded in the same way. XML encoding rules for data nodes are defined in [RFC6020]. JSON encoding rules are defined in [I-D.ietf-netmod-yang-json]. This encoding is valid JSON, but also has special encoding rules to identify module namespaces and provide consistent type processing of YANG data.

3.4.2. On-Change Subscriptions

In an on-change subscription, updates need to allow to differentiate between data nodes that were newly created since the last update, data nodes that were deleted, and data nodes whose value changed.

XML encoding rules correspond to how data would be encoded in input to Netconf edit-config operations as specified in [RFC6241] section 7.2, adding "operation" attributes to elements in the data subtree. Specifically, the following values will be utilized:

- create: The data identified by the element has been added since the last update.
- delete: The data identified by the element has been deleted since the last update.
- merge: The data identified by the element has been changed since the last update.
- replace: The data identified by the element has been replaced with the update contents since the last update.

The remove value will not be utilized.

Contrary to edit-config operations, the data is sent from the server to the client, not from the client to the server, and will not be restricted to configuration data.

JSON encoding rules are roughly analogous to how data would be encoded in input to a YANG-patch operation, as specified in [I-D.ietf-netconf-yang-patch] section 2.2. However, no edit-ids will be needed. Specifically, changes will be grouped under respective "operation" containers for creations, deletions, and modifications.
3.5. Subscription Filters

Subscriptions can specify filters for subscribed data. The following filters are supported:

- **subtree-filter**: A subtree filter specifies a subtree that the subscription refers to. When specified, updates will only concern data nodes from this subtree. Syntax and semantics correspond to that specified for [RFC6241] section 6.

- **xpath-filter**: An XPath filter specifies an XPath expression applied to the data in an update, assuming XML-encoded data.

If multiple subscription filters are specified, all of them are applied. In other words, it is possible to (for example) apply an XPath filter on top of a subtree filter.

It is conceivable for implementations to support other filters. For example, an on-change filter might specify that changes in values should be sent only when the magnitude of the change since previous updates exceeds a certain threshold. It is possible to augment the subscription data model with additional filter types.

3.6. Push Data Stream and Transport Mapping

Pushing data based on a subscription could be considered analogous to a response to a data retrieval request, e.g. a "get" request. However, contrary to such a request, multiple responses to the same request may get sent over a longer period of time.

A more suitable mechanism is therefore that of a notification. Contrary to notifications associated with alarms and unexpected event occurrences, push updates are solicited, i.e. tied to a particular subscription which triggered the notification. (An alternative conceptual model would consider a subscription an "opt-in" filter on a continuous stream of updates.)

The notification contains several parameters:

- A subscription correlator, referencing the name of the subscription on whose behalf the notification is sent.

- A data node that contains a representation of the datastore subtree containing the updates. The subtree is filtered per access control rules to contain only data that the subscriber is authorized to see. Also, depending on the subscription type, i.e., specifically for on-change subscriptions, the subtree contains only the data nodes that contain actual changes. (This
can be simply a node of type string or, for XML-based encoding, anyxml.)

Notifications are sent using `<notification>` elements as defined in [RFC5277]. Alternative transports are conceivable but outside the scope of this specification.

The solution specified in this document uses notifications to communicate datastore updates. The contents of the notification includes a set of explicitly defined data nodes. For this purpose, two new generic notifications are introduced, "push-update" and "push-change-update". Both notifications are used to define how to carry data records with updates of datastore contents as specified by a subscription.

Push-update notification defines updates for a periodic subscription, as well as for the initial update of an on-change subscription used to synchronize the receiver at the start of a new subscription. The update record contains a data snippet that contains an instantiated subtree with the subscribed contents. The content of the update record is equivalent to the contents that would be obtained had the same data been explicitly retrieved using e.g. a Netconf "get"-operation, with the same filters applied.

The contents of the notification conceptually represents the union of all data nodes in the yang modules supported by the server. However, in a YANG data model, it is not practical to model the precise data contained in the updates as part of the notification. This is because the specific data nodes supported depend on the implementing system and may even vary dynamically. Therefore, to capture this data, a single parameter that can represent any datastore contents is used, not parameters that represent data nodes one at a time.

Push-change-update notification defines updates for on-change subscriptions. The update record here contains a data snippet that indicates the changes that data nodes have undergone, i.e. that indicates which data nodes have been created, deleted, or had changes to their values. The format follows the same format that operations that apply changes to a data tree would apply, indicating the creates, deletes, and modifications of data nodes.

The following is an example of push notification. It contains an update for subscription my-sub, including a subtree with root foo that contains a leaf, bar:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<notification>
  <foo>
    <bar/>
  </foo>
</notification>
```
The following is an example of an on-change notification. It contains an update for subscription my-on-change-sub, including a new value for a leaf called beta, which is a child of a top-level container called alpha:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<notification
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <subscription-id
        xmlns="urn:ietf:params:xml:ns:netconf:datstore-push:1.0">
        my-on-change-sub
    </subscription-id>
    <eventTime>2015-10-13T12:13:02Z</eventTime>
    <datastore-changes-xml xmlns="urn:ietf:params:xml:ns:netconf:datstore-push:1.0">
        <alpha xmlns="http://example.com/yang-push/1.0">
            <beta>1500</beta>
        </alpha>
    </datastore-changes-xml>
</notification>
```

Figure 2: Push example for on change

The equivalent update when requesting json encoding:
<?xml version="1.0" encoding="UTF-8"?>
<notification
   xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
   <subscription-id xmlns="urn:ietf:params:xml:ns:netconf:datastore-push:1.0">
      my-on-change-sub
   </subscription-id>
   <eventTime>2015-10-13T12:13:02Z</eventTime>
   <datastore-changes-json
      xmlns="urn:ietf:params:xml:ns:netconf:datastore-push:1.0">
      {
         "ietf-yang-patch:yang-patch": {
            "patch-id": [
               null
            ],
            "edit": [
               {
                  "edit-id": "edit1",
                  "operation": "merge",
                  "target": "/alpha/beta",
                  "value": {
                     "beta": 1500
                  }
               }
            ]
         }
      }
   </datastore-changes-json>
</notification>

Figure 3: Push example for on change with JSON

When the beta leaf is deleted, the server may send
<notification
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <subscription-id xmlns="urn:ietf:params:xml:ns:netconf:datastore-push:1.0">
    my-on-change-sub
  </subscription-id>
  <eventTime>2015-10-13T12:13:02Z</eventTime>
  <datastore-changes-xml xmlns="urn:ietf:params:xml:ns:netconf:datastore-push:1.0">
    <alpha xmlns="http://example.com/yang-push/1.0">
      <beta xc:operation="delete"/>
    </alpha>
  </datastore-changes-xml>
</notification>

3.7. Subscription management

There are two ways in which subscriptions can be managed: RPC-based, and configuration based.

3.7.1. Subscription management by RPC

RPC-based subscription allows a subscriber to create a subscription via an RPC call. The subscriber and the receiver are the same entity, i.e. a subscriber cannot subscribe or in other ways interfere with a subscription on another receiver’s behalf. The lifecycle of the subscription is dependent on the lifecycle of the transport session over which the subscription was requested. For example, when a Netconf session over which a subscription was created is torn down, the subscription is automatically terminated (and needs to be re-initiated when a new session is established). Alternatively, a subscriber can also decide to delete a subscription via another RPC.

When a create-subscription request is successful, the subscription identifier of the freshly created subscription is returned.

A subscription can be rejected for multiple reasons, including the lack of authorization to create a subscription, the lack of read authorization on the requested data node, or the inability of the server to provide a stream with the requested semantics. Rejections trigger the generation of an rpc-reply with an rpc-error element, which indicates why the subscription was rejected and, possibly, negotiation information to facilitate the generation of subscription requests that can be served. The contents of the rpc-error element
follow the specification in [RFC6241]. Datastore-push-specific content is included under <error-info>. When the requester is not authorized to read the requested data node, the returned <error-info> indicates an authorization error and the requested node. For instance, for the following request:

```
<netconf:rpc message-id="101"
 xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
 <create-subscription
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <stream>push-update</stream>
  <filter netconf:type="xpath"
   xmlns:ex="http://example.com/foo/1.0"
   select="/ex:foo"/>
  <period xmlns="urn:ietf:params:xml:ns:netconf:datastore-push:1.0">500</period>
  <encoding
   xmlns="urn:ietf:params:xml:ns:netconf:datastore-push:1.0">
   encode-xml
  </encoding>
 </create-subscription>
</netconf:rpc>
```

Figure 5: Create-Subscription example

the server may return:

```
<rpc-reply message-id="101"
 xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
 <rpc-error>
  <error-type>application</error-type>
  <error-tag>access-denied</error-tag>
  <error-severity>error</error-severity>
  <error-info>
   <access-denied xmlns="urn:ietf:params:xml:ns:netconf:datastore-push:1.0">
    <data-node>/ex:foo</data-node>
   </access-denied>
  </error-info>
 </rpc-error>
</rpc-reply>
```

Figure 6: Error response example

When the requester is not authorized to execute a subscription request, no <error-info> element should be included in the response.
3.7.2. Subscription management by configuration

Configuration-based subscription allows a subscription to be established as part of a server’s configuration. This allows to persist subscriptions. As part of a configured subscription, a receiver needs to be specified. It is thus possible to have a different system acting as subscriber (the client creating the subscription) and as receiver (the client receiving the updates).

3.8. Other considerations

3.8.1. Authorization

A receiver of subscription data may only be sent updates for which they have proper authorization. Data that is being pushed therefore needs to be subjected to a filter that applies all corresponding rules applicable at the time of a specific pushed update, removing any non-authorized data as applicable.

The authorization model for data in YANG datastores is described in the Netconf Access Control Model [RFC6536]. However, some clarifications to that RFC are needed so that the desired access control behavior is applied to pushed updates.

One of these clarifications is that a subscription may only be established if the Receiver has read access to the target data node.

```
+-------------+                 +-------------+
| subscription|  protocol   |                 |   target    |
| request -->  |  operation  | ------------->  |  data node  |
|             |  allowed?   |   datastore     |  access    |
|             | +-------------+   or state      |  allowed?   |
|             |                 |   data access   | +-------------+
```

Figure 7: Access control for subscription

Likewise if a receiver no longer has read access permission to a target data node, the subscription must be abnormally terminated (with loss of access permission as the reason provided).

Another clarification to [RFC6536] is that each of the individual nodes in a pushed update must also go through access control filtering. This includes new nodes added since the last push update, as well as existing nodes. For each of these read access must be verified. The methods of doing this efficiently are left to implementation.
Figure 8: Access control for push updates

If there are read access control changes applied under the target node, no notifications indicating the fact that this has occurred need to be provided.

3.8.2. Additional subscription primitives

Other possible operations include the ability for a Subscriber to request the suspension/resumption of a Subscription with a Publisher. However, subscriber driven suspension is not viewed as essential at this time, as a simpler alternative is to remove a subscription and recreate it when needed.

It should be noted that this does not affect the ability of the Publisher to suspend a subscription. This can occur in cases the server is not able to serve the subscription for a certain period of time, and indicated by a corresponding notification.

3.8.3. Robustness and reliability considerations

Particularly in the case of on-change push updates, it is important that push updates do not get lost. However, datastore-push uses a secure and reliable transport. Notifications are not getting reordered, and in addition contain a time stamp. For those reasons, we believe that additional reliability mechanisms at the application level, such as sequence numbers for push updates, are not required.

3.8.4. Update size and fragmentation considerations

Depending on the subscription, the volume of updates can become quite large. There is no inherent limitation to the amount of data that can be included in a notification. That said, it may not always be practical to send the entire update in a single chunk. Implementations MAY therefore choose, at their discretion, to "chunk" updates and break them out into several update notifications.

3.8.5. Additional data streams

The conceptual data stream introduced in this specification, datastore-push, includes the entire YANG datastore in its scope. It...
is conceivable to introduce other data streams with more limited scope, for example:

- operdata-push, a datastream containing all operational (read-only) data of a YANG datastore
- operdata-nocounts-push, a datastream containing all operational (read-only) data with the exception of counters

Those data streams make particular sense for use cases involving service assurance (not relying on operational data), and for use cases requiring on-change update triggers which make no sense to support in conjunction with fast-changing counters. While it is possible to specify subtree filters on datastore-push to the same effect, having those data streams greatly simplifies articulating subscriptions in such scenarios.

3.8.6. Implementation considerations

Implementation specifics are outside the scope of this specification. That said, it should be noted that monitoring of operational state changes inside a system can be associated with significant implementation challenges.

Even periodic retrieval of operational state alone, to be able to push it, can consume considerable system resources. Configuration data may in many cases be persisted in an actual database or a configuration file, where retrieval of the database content or the file itself is reasonably straightforward and computationally inexpensive. However, retrieval of operational data may, depending on the implementation, require invocation of APIs, possibly on an object-by-object basis, possibly involving additional internal interrupts, etc.

For those reasons, if is important for an implementation to understand what subscriptions it can or cannot support. It is far preferrable to decline a subscription request, than to accept it only to result in subsequent failure later.

Whether or not a subscription can be supported will in general be determined by a combination of several factors, including the subscription policy (on-change or periodic, with on-change in general being the more challenging of the two), the period in which to report changes (1 second periods will consume more resources than 1 hour periods), the amount of data in the subtree that is being subscribed to, and the number and combination of other subscriptions that are concurrently being serviced.
When providing access control to every node in a pushed update, it is possible to make and update efficient access control filters for an update. These filters can be set upon subscription and applied against a stream of updates. These filters need only be updated when (a) there is a new node added/removed from the subscribed tree with different permissions than its parent, or (b) read access permissions have been changed on nodes under the target node for the subscriber.

4. A YANG data model for management of datastore push subscriptions

4.1. Overview

The YANG data model for datastore push subscriptions is depicted in the following figure.

```
module: ietf-datastore-push
  +--ro system-streams
    |  +--ro system-stream* system-stream
  +--rw filters
    |  +--rw filter* [filter-id]
    |      |  +--rw filter-id filter-id
    |      |      +--:(subtree)
    |      |         |  +--rw subtree-filter? subtree-filter
    |      |         +--:(xpath)
    |      |           |  +--rw xpath-filter? yang:xpath1.0
  +--rw subscription-config
    |  +--rw datastore-push-subscription* [subscription-id]
    |      |  +--rw subscription-id subscription-id
    |      |  +--rw target-datastore? datastore
    |      |  +--rw stream? system-stream
    |      |  +--rw encoding? encoding
    |      |  +--rw start-time? yang:date-and-time
    |      |  +--rw stop-time? yang:date-and-time
    |      |      +--rw (update-trigger)?
    |      |         |  +--:(periodic)
    |      |             |  +--rw period? yang:timeticks
    |      |             |  +--:(on-change)
    |      |             |     +--rw no-synch-on-start? empty
    |      |             |     +--rw dampening-period yang:timeticks
    |      |             |     +--rw excluded-change* change-type
    |      |      +--rw (filterspec)?
    |      |         |  +--:(inline)
    |      |             |  +--rw (filter-type)?
    |      |             |  |  +--:(subtree)
    |      |             |  |     +--rw subtree-filter? subtree-filter
    |      |             |  |     +--:(xpath)
    |      |             |  |       |  +--rw xpath-filter? yang:xpath1.0
```
The components of the model are described in the following subsections.
4.2. System streams

Container "system-streams" is used to indicate which data streams are provided by the system and can be subscribed to. For this purpose, it contains a leaf list of data nodes identifying the supported streams.

4.3. Filters

Container "filters" contains a list of configurable data filters, each specified in its own list element. This allows users to configure filters separately from an actual subscription, which can then be referenced from a subscription. This facilitates the reuse of filter definitions, which can be important in case of complex filter conditions.

Two types of filters can be specified as part of a filter list element. Subtree filters follow syntax and semantics of RFC 6241 and allow to specify which subtree(s) to subscribe to. In addition, XPath filters can be specified for more complex filter conditions. If several filters are specified, when both types of filters are included, a logical "and" applies.

It is conceivable to introduce other types of filters; in that case, the data model needs to be augmented accordingly.

4.4. Subscription configuration

Container "subscription-config" allows for the static configuration of subscriptions, i.e. subscriptions that are created via configuration as opposed to RPC. Each subscription is represented through its own list element, including the following components:

- "subscription-id" is an identifier used to refer to the subscription.
- "target-datastore" is used to refer to the datastore the subscription refer to. By default, the datastore will always be "running".
- "stream" refers to the stream being subscribed to. The subscription model assumes the presence of perpetual and continuous streams of updates. Various streams are defined: "push-update" covers the entire set of YANG data in the server. "operational-push" covers all operational data, while "config-push" covers all configuration data. Other streams could be introduced in augmentations to the model by introducing additional identities.
o "encoding" refers to the encoding requested for the data updates. By default, updates are encoded using XML. However, JSON can be requested as an option and other encodings may be supported in the future.

o "start-time" specifies when the subscription is supposed to start. The start time also serves as anchor time for periodic subscriptions (see below).

o "stop-time" specifies a stop time for the subscription. Once the stop time is reached, the subscription is automatically terminated. However, even when terminated, the subscription entry remains part of the configuration unless explicitly deleted from the configuration. It is possible to effectively "resume" a stopped subscription by reconfiguring the stop time.

o A choice of subscription policies allows to define when to send new updates - periodic or on change.

* For periodic subscriptions, the trigger is defined by a "period", a parameter that defines the interval with which updates are to be pushed. The start time of the subscription serves as anchor time, defining one specific point in time at which an update needs to be sent. Update intervals always fall on the points in time that are a multiple of a period after the start time.

* For on-change subscriptions, the trigger occurs whenever a change in the subscribed information is detected. On-change subscriptions have more complex semantics that is guided by additional parameters. "dampening-period" specifies the interval that must pass before a successive update for the same data node is sent. The first time a change is detected, the update is sent immediately. If a subsequent change is detected, another update is only sent once the dampening period has passed, containing the value of the data node that is then valid. "excluded-change" allows to restrict the types of changes for which updates are sent (changes to object values, object creation or deletion events). "no-synch-on-start" is a flag that allows to specify whether or not a complete update with all the subscribed data should be sent at the beginning of a subscription; if the flag is omitted, a complete update is sent to facilitate synchronization. It is conceivable to augment the data model with additional parameters in the future to specify even more refined policies, such as parameters that specify the magnitude of a change that must occur before an update is triggered.
Filters for a subscription can be specified using a choice, allowing to either reference a filter that has been separately configured or entering its definition inline.

Finally, a receiver for the subscription can be specified. The receiver does not have to be the same system that configures the subscription.

It should be noted that a subscription created through configuration cannot be deleted using an RPC. Likewise, subscriptions created through RPC cannot be deleted through configuration.

4.5. Subscription monitoring

Subscriptions can be subjected to management themselves. For example, it is possible that a server may no longer be able to serve a subscription that it had previously accepted. Perhaps it has run out of resources, or internal errors may have occurred. When this is the case, a server needs to be able to temporarily suspend the subscription, or even to terminate it. More generally, the server should provide a means by which the status of subscriptions can be monitored.

Container "subscriptions", contains operational data for all subscriptions that are currently active. This includes subscriptions that were created using RPC, as well as subscriptions created as part of the configuration when current time is between start and stop time.

Each subscription is represented as a list element "datastore-push-subscription". The associated information includes an identifier for the subscription, a subscription status, as well as the various subscription parameters that are in effect. The subscription status indicates whether the subscription is currently active and healthy, or if it is degraded in some form.

Subscriptions are automatically removed from the list once they expire (reaching stop-time) or are terminated, whether through RPC or deletion from the configuration.

4.6. Notifications

A server needs to indicate any changes in status of a subscription to the receiver through a notification. Specifically, subscribers need to be informed of the following:

- A subscription has been temporarily suspended (including the reason)
A subscription (that had been suspended earlier) is once again operational

A subscription has been terminated (including the reason)

A subscription has been modified (including the current set of subscription parameters in effect)

Finally, a server might provide additional information about subscriptions, such as statistics about the number of data updates that were sent. However, such information is currently outside the scope of this specification.

4.7. RPCs

Yang-push subscriptions are created, modified, and deleted using three RPCs.

4.7.1. Create-subscription RPC

The subscriber sends a create-subscription RPC with the parameters in section 3.1. For instance

```
<netconf:rpc message-id="101"
    xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <create-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <stream>push-update</stream>
    <filter netconf:type="xpath"
      xmlns:ex="http://example.com/foo/1.0"
      select="/ex:foo"/>
    <period xmlns="urn:ietf:params:xml:ns:netconf:
      datastore-push:1.0">500</period>
    <encoding xmlns="urn:ietf:params:xml:ns:netconf:
      datastore-push:1.0">encode-xml</encoding>
  </create-subscription>
</netconf:rpc>
```

Figure 10: Create-subscription RPC

The server must respond explicitly positively (i.e., subscription accepted) or negatively (i.e., subscription rejected) to the request. Positive responses include the subscription-id of the accepted subscription. In that case a server may respond:
A subscription can be rejected for multiple reasons, including the lack of authorization to create a subscription, the lack of read authorization on the requested data node, or the inability of the server to provide a stream with the requested semantics. Rejections trigger the generation of an rpc-reply with an rpc-error element, which indicates why the subscription was rejected and, possibly, negotiation information to facilitate the generation of subscription requests that can be served. The contents of the rpc-error element follow the specification in [RFC6241]. Datastore-push-specific content is included under <error-info>.

When the requester is not authorized to read the requested data node, the returned <error-info> indicates an authorization error and the requested node. For instance, if the above request was unauthorized to read node "ex:foo" the server may return:

```xml
<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <error-type>application</error-type>
    <error-tag>access-denied</error-tag>
    <error-severity>error</error-severity>
    <error-info>
        <access-denied xmlns="urn:ietf:params:xml:ns:netconf:datastore-push:1.0">
            <data-node>/ex:foo</data-node>
        </access-denied>
    </error-info>
</rpc-reply>
```

Figure 12: Create-subscription access denied response
When the requester is not authorized to execute a subscription request, no `<error-info>` element should be included in the response. If a request is rejected because the server is not capable to serve it, the server SHOULD include in the returned error what request parameters were not supported and what subscription parameters would have been accepted for the request. This information is included in the `<error-info>`, which is split into two sections. First, `<unsupported-parameters>`, which includes the parameters in the request the server cannot serve. Second `<supported-subscription>`, which constitute suggestions that, when followed, increase the likelihood of success for subsequent requests. However, they are no guarantee that subsequent requests for this client or others will in fact be accepted.

For example, for the following request:

```xml
<netconf:rpc message-id="101"
 xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <create-subscription
      xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <stream>push-update</stream>
    <filter netconf:type="xpath"
      xmlns:ex="http://example.com/foo/1.0"
      select="/ex:foo="/>
    <dampening-period
      xmlns="urn:ietf:params:xml:ns:netconf:datastore-push:1.0">
      10
    </dampening-period>
    <encoding
      xmlns="urn:ietf:params:xml:ns:netconf:datastore-push:1.0">
      encode-xml
    </encoding>
  </create-subscription>
</netconf:rpc>
```

Figure 13: Create-subscription request example 2

A server that cannot serve on-change updates may return the following:
Figure 14: Create-subscription error response example 2

4.7.2. Modify-subscription RPC

The subscriber may send a modify-subscription for a previously accepted subscription that has not been deleted. The subscriber may change any subscription parameters by including the new values in the modify-subscription rpc. Parameters not included in the rpc should remain unmodified. For illustration purposes we include an exchange example where a subscriber modifies the period of the subscription.
Figure 15: Modify subscription request

The server must respond explicitly positively (i.e., subscription accepted) or negatively (i.e., subscription rejected) to the request. Positive responses include the subscription-id of the accepted subscription. In that case a server may respond:
If the subscription modification is rejected, the server must send a response like it does for a create-subscription and maintain the subscription as it was before the modification request. A subscription may be modified multiple times.
4.7.3. Delete-subscription RPC

To stop receiving updates from a subscription and effectively eliminate the subscription, it can send a delete-subscription RPC, which takes as only input the subscription-id. For example

```xml
<netconf:rpc message-id="103"
     xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <delete-subscription
     xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <subscription-id xmlns="urn:ietf:params:xml:ns:netconf:
                      datastore-push:1.0">
      my-sub
    </subscription-id>
  </delete-subscription>
</netconf:rpc>

<rpc-reply message-id="103"
            xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Figure 17: Delete subscription

5. YANG module

```xml
<CODE BEGINS>
file "ietf-datastore-push@2015-10-15.yang"

module ietf-datastore-push {
  prefix yp;

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

  organization "IETF";
  contact
  "WG Web:  <http://tools.ietf.org/wg/netconf/>
   WG List:  <mailto:netconf@ietf.org>
   WG Chair: Mahesh Jethanandani
             <mailto:mjethanandani@gmail.com>
```
This module contains conceptual YANG specifications for datastore push.

revision 2015-10-15 {
    description
        "Initial revision."
    reference
        "YANG Datastore Push, draft-ietf-netconf-yang-push-00"
}

feature on-change {
    description
        "This feature indicates that on-change updates are supported."
}

feature json {
    description
        "This feature indicates that JSON encoding of push updates is supported."
}

identity subscription-stream-status {
    description
        "Base identity for the status of subscriptions and datastreams."
}

identity active {
    base subscription-stream-status;
    description
        "Status is active and healthy."
}

identity inactive {
    base subscription-stream-status;
    description
"Status is inactive, for example outside the interval between start time and stop time."
)

identity in-error {
  base subscription-stream-status;
  description
    "The status is in error or degraded, meaning that stream and/or subscription are currently unable to provide the negotiated updates."
}

identity subscription-errors {
  description
    "Base identity for subscription errors."
}

identity internal-error {
  base subscription-errors;
  description
    "Subscription failures caused by server internal error."
}

identity no-resources {
  base subscription-errors;
  description
    "Lack of resources, e.g. CPU, memory, bandwidth"
}

identity subscription-deleted {
  base subscription-errors;
  description
    "The subscription was terminated because the subscription was deleted."
}

identity other {
  base subscription-errors;
  description
    "Fallback reason - any other reason"
}

identity encodings {
  description
    "Base identity to represent data encodings"
}

identity encode-xml {
  description
    "Data encoding encoding-xml"
}
base encodings;
description
"Encode data using XML";
}

identity encode-json {
    base encodings;
description
"Encode data using JSON";
}

identity system-streams {
description
"Base identity to represent a conceptual system-provided
datastream of datastore updates with predefined semantics.";
}

identity datastore-push {
    base system-streams;
description
"A conceptual datastream consisting of all datastore
updates, including operational and configuration data.";
}

identity operational-push {
    base system-streams;
description
"A conceptual datastream consisting of updates of all
operational data.";
}

identity config-push {
    base system-streams;
description
"A conceptual datastream consisting of updates of all
configuration data.";
}

identity datastore {
description
"An identity that represents a datastore.";
}

identity running {
    base datastore;
description
"Designates the running datastore";
}
identity startup {
    base datastore;
    description
        "Designates the startup datastore";
}

typedef datastore-contents-xml {
    type string;
    description
        "This type is used to represent datastore contents,
        i.e. a set of data nodes with their values, in XML.
        The syntax corresponds to the syntax of the data payload
        returned in a corresponding Netconf get operation with the
        same filter parameters applied."
        reference "RFC 6241 section 7.7";
}

typedef datastore-changes-xml {
    type string;
    description
        "This type is used to represent a set of changes in a
        datastore encoded in XML, indicating for datanodes whether
        they have been created, deleted, or updated. The syntax
        corresponds to the syntax used to when editing a
        datastore using the edit-config operation in Netconf."
        reference "RFC 6241 section 7.2";
}

typedef datastore-contents-json {
    type string;
    description
        "This type is used to represent datastore contents,
        i.e. a set of data nodes with their values, in JSON.
        The syntax corresponds to the syntax of the data
        payload returned in a corresponding RESTCONF get
        operation with the same filter parameters applied."
        reference "RESTCONF Protocol";
}

typedef datastore-changes-json {
    type string;
    description
        "This type is used to represent a set of changes in a
        datastore encoded in JSON, indicating for datanodes whether
        they have been created, deleted, or updated. The syntax
        corresponds to the syntax used to patch a datastore
        using the yang-patch operation with Restconf."
        reference "draft-ietf-netconf-yang-patch";
typedef filter-id {
  type string;
  description
    "This type defines an identifier for a filter.";
}

typedef subtree-filter {
  type string;
  description
    "This type is used to specify the subtree that the
    subscription refers to. Its syntax follows the subtree
    filter syntax specified for Netconf in RFC 6241,
    section 6.";
    reference "RFC 6241 section 6";
}

typedef datastore {
  type identityref {
    base datastore;
  }
  description
    "Used to refer to a datastore, for example, to running";
}

typedef subscription-id {
  type string {
    length "1 .. max";
  }
  description
    "A client-provided identifier for the subscription.";
}

typedef subscription-term-reason {
  type identityref {
    base subscription-errors;
  }
  description
    "Reason for a server to terminate a subscription.";
}

typedef subscription-susp-reason {
  type identityref {
    base subscription-errors;
  }
  description
    "Reason for a server to suspend a subscription.";
}
typedef encoding {
    type identityref {
        base encodings;
    }
    description
    "Specifies a data encoding, e.g. for a data subscription.";
}

typedef change-type {
    type enumeration {
        enum "create" {
            description
            "A new data node was created";
        }
        enum "delete" {
            description
            "A data node was deleted";
        }
        enum "modify" {
            description
            "The value of a data node has changed";
        }
    }
    description
    "Specifies different types of changes that may occur to a datastore.";
}

typedef system-stream {
    type identityref {
        base system-streams;
    }
    description
    "Specifies a system-provided datastream.";
}

typedef filter-ref {
    type leafref {
        path "/yp:filters/yp:filter/yp:filter-id";
    }
    description
    "This type is used to reference a yang push filter.";
}

grouping datatree-filter {
    description

"This grouping defines filters for a datastore tree.";
choice filter-type {
  description
  "A filter needs to be a single filter of a given type. Mixing and matching of multiple filters does not occur at the level of this grouping.";
  case subtree {
    description
    "Subtree filter";
    leaf subtree-filter {
      type subtree-filter;
      description
      "Datastore subtree of interest.";
    }
  }
  case xpath {
    description
    "XPath filter";
    leaf xpath-filter {
      type yang:xpath1.0;
      description
      "Xpath defining the data items of interest.";
    }
  }
}
}

grouping subscription-info {
  description
  "This grouping describes basic information concerning a subscription.";
  leaf target-datastore {
    type datastore;
    default "running";
    description
    "The datastore that is the target of the subscription. If not specified, running applies.";
  }
  leaf stream {
    type system-stream;
    default "datastore-push";
    description
    "The name of the stream subscribed to.";
  }
  leaf encoding {
    type encoding;
    default "encode-xml";
    description
"The type of encoding for the subscribed data. Default is XML."
}
leaf start-time {
type yang:date-and-time;

description
"Designates the time at which a subscription is supposed to start, or immediately, in case the start-time is in the past. For periodic subscription, the start time also serves as anchor time from which the time of the next update is computed. The next update will take place at the next period interval from the anchor time. For example, for an anchor time at the top of a minute and a period interval of a minute, the next update will be sent at the top of the next minute."
}
leaf stop-time {
type yang:date-and-time;

description
"Designates the time at which a subscription will end. When a subscription reaches its stop time, it will be automatically deleted."
}
choice update-trigger {
description
"Defines necessary conditions for sending an event to the subscriber."

case periodic {
description
"The agent is requested to notify periodically the current values of the datastore or the subset defined by the filter."
leaf period {
type yang:timeticks;

description
"Elapsed time between notifications."
}
}

case on-change {
description
"The agent is requested to notify changes in values in the datastore or a subset of it defined by a filter."
leaf no-synch-on-start {
type empty;

description
"This leaf acts as a flag that determines behavior at the start of the subscription. When present,
synchronization of state at the beginning of the subscription is outside the scope of the subscription. Only updates about changes that are observed from the start time, i.e. only push-change-update notifications are sent.
When absent (default behavior), in order to facilitate a receiver’s synchronization, a full update is sent when the subscription starts using a push-update notification, just like in the case of a periodic subscription. After that, push-change-update notifications are sent."

leaf dampening-period {
  type yang:timeticks;
  mandatory true;
  description
  "Minimum amount of time that needs to have passed since the last time an update was provided.";
}

leaf-list excluded-change {
  type change-type;
  description
  "Use to restrict which changes trigger an update. For example, if modify is excluded, only creation and deletion of objects is reported.";
}

choice filterspec {
  description
  "Filter can be specified in-line, as part of the subscription, or configured separately and referenced here. If no filter is specified, the entire datatree is of interest.";
  case inline {
    description
    "Filter is defined as part of the subscription.";
    uses datatree-filter;
  }
  case by-reference {
    description
    "Incorporate a filter that has been configured separately.";
    leaf filter-ref {
      type filter-ref;
      description
      "References the filter to incorporate for the
subscription.

grouping receiver-info {
  description "Defines a reusable snippet that defines the address of the
intended receiver of push updates for a subscription.";
  container receiver-address {
    description "This container contains the address information of the
receiver.";
    choice push-base-transport {
      description "This choice can be augmented with different options,
depending on the transport underlying the push
transport.";
      case tcpudp {
        description "For Netconf and Restconf, TCP is the base transport.";
        container tcpudp {
          description "Contains TCP / UDP addressing information";
          leaf address {
            type inet:host;
            description "The leaf uniquely specifies the address of the
remote host. One of the following must be
specified: an ipv4 address, an ipv6 address,
or a host name.";
          }
          leaf port {
            type inet:port-number;
            description "This leaf specifies the port number used to
deliver messages to the remote server.";
          }
        }
      }
    }
  }
}

rpc create-subscription {
  description "This RPC allows a subscriber to create a subscription
on its own behalf. If successful, the subscription remains in effect for the duration of the subscriber’s association with the publisher, or until the subscription is terminated by virtue of a delete-subscription request.

```
input {
  uses subscription-info;
}
```

```
output {
  leaf subscription-id {
    type subscription-id;
    description
    "Identifier used for this subscription."
  }
}
```

```
rpc modify-subscription {
  description
  "This RPC allows a subscriber to modify a subscription that was previously created using create-subscription. If successful, the subscription remains in effect for the duration of the subscriber’s association with the publisher, or until the subscription is terminated by virtue of a delete-subscription request."
  input {
    leaf subscription-id {
      type subscription-id;
      description
      "Identifier to use for this subscription."
    }
  }
}
```

```
rpc delete-subscription {
  description
  "This RPC allows a subscriber to delete a subscription that was previously created using create-subscription."
  input {
    leaf subscription-id {
      type subscription-id;
      description
      "Identifier of the subscription that is to be deleted. Only subscriptions that were created using create-subscription can be deleted via this RPC."
    }
  }
}
```

```
notification push-update {
  description
  "This notification contains a periodic push update.
  
  Clemm, et al. Expires April 17, 2016 [Page 43]"
This notification shall only be sent to receivers of a subscription; it does not constitute a general-purpose notification.

leaf subscription-id {
  type subscription-id;
  mandatory true;
  description "This references the subscription because of which the notification is sent.";
}

leaf time-of-update {
  type yang:date-and-time;
  description "This leaf contains the time of the update.";
}

choice encoding {
  description "Distinguish between the proper encoding that was specified for the subscription";
  case encode-xml {
    description "XML encoding";
    leaf datastore-contents-xml {
      type datastore-contents-xml;
      description "This contains data encoded in XML, per the subscription.";
    }
  }
  case encode-json {
    if-feature json;
    description "JSON encoding";
    leaf datastore-contents-json {
      type datastore-contents-json;
      description "This leaf contains data encoded in JSON, per the subscription.";
    }
  }
}

notification push-change-update {
  description "This notification contains an on-change push update. This notification shall only be sent to the receivers of a subscription; it does not constitute a general-purpose notification.";
}
leaf subscription-id {
  type subscription-id;
  mandatory true;
  description
    "This references the subscription because of which the
     notification is sent.";
}

leaf time-of-update {
  type yang:date-and-time;
  description
    "This leaf contains the time of the update, i.e. the
     time at which the change was observed.";
}

choice encoding {
  description
    "Distinguish between the proper encoding that was specified
     for the subscription";
  case encode-xml {
    description
      "XML encoding";
    leaf datastore-changes-xml {
      type datastore-changes-xml;
      description
        "This contains datastore contents that has changed
         since the previous update, per the terms of the
         subscription. Changes are encoded analogous to
         the syntax of a corresponding Netconf edit-config
         operation.";
    }
  }
  case encode-json {
    if-feature json;
    description
      "JSON encoding";
    leaf datastore-changes-yang {
      type datastore-changes-yang;
      description
        "This contains datastore contents that has changed
         since the previous update, per the terms of the
         subscription. Changes are encoded analogous
         to the syntax of a corresponding RESTCONF yang-patch
         operation.";
    }
  }
}

notification subscription-started {
  description
"This notification indicates that a subscription has started and data updates are beginning to be sent. This notification shall only be sent to receivers of a subscription; it does not constitute a general-purpose notification.";
leaf subscription-id {
  type subscription-id;
  mandatory true;
  description
    "This references the affected subscription.";
}
uses subscription-info;

notification subscription-suspended {
  description
    "This notification indicates that a suspension of the subscription by the server has occurred. No further datastore updates will be sent until subscription resumes. This notification shall only be sent to receivers of a subscription; it does not constitute a general-purpose notification.";
leaf subscription-id {
  type subscription-id;
  mandatory true;
  description
    "This references the affected subscription.";
}
leaf reason {
  type subscription-susp-reason;
  description
    "Provides a reason for why the subscription was suspended.";
}
}

notification subscription-resumed {
  description
    "This notification indicates that a subscription that had previously been suspended has resumed. Datastore updates will once again be sent.";
leaf subscription-id {
  type subscription-id;
  mandatory true;
  description
    "This references the affected subscription.";
}
}

notification subscription-modified {

description
"This notification indicates that a subscription has been modified. Datastore updates sent from this point on will conform to the modified terms of the subscription."
leaf subscription-id {
    type subscription-id;
    mandatory true;
    description
    "This references the affected subscription.";
}
uses subscription-info;
}
notification subscription-terminated {
    description
    "This notification indicates that a subscription has been terminated."
    leaf subscription-id {
        type subscription-id;
        mandatory true;
        description
        "This references the affected subscription.";
    }
    leaf reason {
        type subscription-term-reason;
        description
        "Provides a reason for why the subscription was terminated.";
    }
}
container system-streams {
config false;
    description
    "This container contains a leaf list of built-in streams that are provided by the system."
    leaf-list system-stream {
        type system-stream;
        description
        "Identifies a built-in stream that is supported by the system. Streams are associated with their own identities, each of which carries a special semantics.";
    }
}
container filters {
    description
    "This container contains a list of configurable filters that can be applied to subscriptions. This facilitates the reuse of complex filters once defined.";
}
list filter {
key "filter-id";
description "A list of configurable filters that can be applied to subscriptions.";
leaf filter-id {
type filter-id;
description "An identifier to differentiate between filters.";
}
uses datatree-filter;
}
}
container subscription-config {
description "Contains the list of subscriptions that are configured, as opposed to established via RPC or other means.";
list datastore-push-subscription {
key "subscription-id";
description "Content of a yang-push subscription.";
leaf subscription-id {
type subscription-id;
description "Identifier to use for this subscription.";
}
uses subscription-info;
uses receiver-info;
}
}
container subscriptions {
config false;
description "Contains the list of currently active subscriptions, i.e. subscriptions that are currently in effect, used for subscription management and monitoring purposes. This includes subscriptions that have been setup via RPC primitives, e.g. create-subscription, delete-subscription, and modify-subscription, as well as subscriptions that have been established via configuration.";
list datastore-push-subscription {
key "subscription-id";
config false;
description "Content of a yang-push subscription. Subscriptions can be created using a control channel or RPC, or be established through configuration.";
leaf subscription-id {

type subscription-id;
  description
    "Identifier of this subscription.";
}
leaf configured-subscription {
  type empty;
  description
    "The presence of this leaf indicates that the subscription originated from configuration, not through a control channel or RPC.";
}
leaf subscription-status {
  type identityref {
    base subscription-stream-status;
  }
  description
    "The status of the subscription.";
}
uses subscription-info;
uses receiver-info;
}
}
</CODE ENDS>

6. Security Considerations

Subscriptions could be used to attempt to overload servers of YANG datastores. For this reason, it is important that the server has the ability to decline a subscription request if it would deplete its resources. In addition, a server needs to be able to suspend an existing subscription when needed. When this occur, the subscription status is updated accordingly and the clients are notified. Likewise, requests for subscriptions need to be properly authorized.

A subscription could be used to retrieve data in subtrees that a client has not authorized access to. Therefore it is important that data pushed based on subscriptions is authorized in the same way that regular data retrieval operations are. Data being pushed to a client needs therefore to be filtered accordingly, just like if the data were being retrieved on-demand. The Netconf Authorization Control Model applies.

A subscription could be configured on another receiver’s behalf, with the goal of flooding that receiver with updates. One or more publishers could be used to overwhelm a receiver which doesn’t even support subscriptions. Clients which do not want pushed data need only terminate or refuse any transport sessions from the publisher.
7. Acknowledgments

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from Ambika Prasad Tripathy and Einar Nilsen-Nygaard.

8. References

8.1. Normative References


8.2. Informative References


[I-D.i2rs-pub-sub-requirements] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", draft-ietf-i2rs-pub-sub-requirements-03 (work in progress), October 2015.
[I-D.ietf-netconf-restconf]

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

[I-D.ietf-netmod-yang-json]
Lhotka, L., "JSON Encoding of Data Modeled with YANG", draft-ietf-netmod-yang-json-06 (work in progress), October 2015.

[I-D.voit-netmod-peer-mount-requirements]

Authors' Addresses

Alexander Clemm
Cisco Systems
EMail: alex@cisco.com

Alberto Gonzalez Prieto
Cisco Systems
EMail: albertgo@cisco.com

Eric Voit
Cisco Systems
EMail: evoit@cisco.com
Zero Touch Provisioning for NETCONF Call Home

draft-ietf-netconf-zerotouch-04

Abstract

This draft presents a technique for establishing a secure NETCONF or
RESTCONF connection between a newly deployed device, configured with
just its factory default settings, and its rightful owner’s network
management system (NMS).

Status of This Memo

This Internet-Draft is submitted in full conformance with the
provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering
Task Force (IETF). Note that other groups may also distribute
working documents as Internet-Drafts. The list of current Internet-
Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months
and may be updated, replaced, or obsoleted by other documents at any
time. It is inappropriate to use Internet-Drafts as reference
material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 21, 2016.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal
Provisions Relating to IETF Documents
(http://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ........................................... 3
   1.1. Use Cases ........................................ 3
   1.2. Terminology ..................................... 4
   1.3. Tree Diagrams ................................... 5
2. Guiding Principles .................................... 6
   2.1. Trust Anchors ................................... 6
   2.2. Conveying Trust .................................. 6
   2.3. Ownership ........................................ 7
   2.4. Information Types ................................ 7
3. Sources for Bootstrapping Data .......................... 8
   3.1. Removable Storage ................................ 8
   3.2. DHCP Server ..................................... 8
   3.3. Internet Based Service ........................... 9
4. Workflow Overview ..................................... 9
   4.1. Onboarding and Ordering Devices ................... 9
   4.2. Owner Stages the Network for Bootstrap ........... 11
   4.3. Device Powers On ................................ 13
5. Device Details ......................................... 16
   5.1. Factory Default State ............................. 16
   5.2. Boot Sequence ................................... 17
   5.3. Validating Signed Data ............................ 19
   5.4. Processing Bootstrap Data ........................ 19
6. YANG-defined API and Artifacts .......................... 20
   6.1. Module Overview .................................. 20
   6.2. API Examples ...................................... 22
      6.2.1. Unsigned Redirect Information ................ 22
      6.2.2. Signed Redirect Information .................. 23
      6.2.3. Unsigned Bootstrap Information ............... 26
      6.2.4. Signed Bootstrap Information ................ 28
      6.2.5. Progress Notifications ....................... 31
   6.3. Artifact Examples ................................ 32
      6.3.1. Signed Redirect Information .................. 32
      6.3.2. Signed Bootstrap Information ................ 33
      6.3.3. Owner Certificate ............................ 35
      6.3.4. Ownership Voucher ............................ 36
   6.4. YANG Module ....................................... 37
7. Security Considerations ............................... 44
   7.1. Immutable storage for trust anchors ............... 44
   7.2. Real time clock .................................. 44
   7.3. Entropy loss over time ............................ 44
   7.4. Serial Numbers ................................... 44
8. IANA Considerations .................................... 45
   8.1. Zero Touch Information DHCP Options ............... 45
1. Introduction

A fundamental business requirement is to reduce costs where possible. For network operators, deploying devices to many locations can be a significant cost, as sending trained specialists to each site to do installations is both cost prohibitive and does not scale.

This document defines bootstrapping strategies enabling a device to securely obtain bootstrapping data with no installer input beyond racking the device and applying power. This bootstrapping data directs the device to install a boot image and an initial configuration, which enables the establishment of a NETCONF [RFC6241] or RESTCONF [draft-ietf-netconf-restconf] connection to its rightful owner’s network management system (NMS).

In order to establish a NETCONF or RESTCONF connection, the initial configuration should include settings such as enabling the NETCONF/RESTCONF service, including parameters needed to support an NMS-initiated or device-initiated connection, and configuring a local administrator account. Examples used in this draft illustrate this using models defined by [RFC7317] and [draft-ietf-netconf-server-model].

1.1. Use Cases

- Connecting to a remotely administered network

  This use-case involves scenarios, such as a remote branch office or convenience store, whereby a device connects as an access gateway to an ISP’s network. Assuming it is not possible to customize the ISP’s network, and with no other nearby device to leverage, the device has no recourse but to

reach out to the public Internet for a well-known service it can bootstrap off of.

- Connecting to a locally administered network

This use-case covers all other scenarios and differs only in that the device may additionally leverage nearby devices, which may direct it to use a local service to bootstrap off of. If no such site-specific information is available, or the device is unable to use the information provided, it can then reach out to network just as it would for the remotely administered network use-case.

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in the sections below are to be interpreted as described in RFC 2119 [RFC2119].

This document defines the following terms:

Artifact: The term "artifact" is used throughout to represent bootstrapping data that can be encoded outside of the RESTCONF protocol. For example, an artifact may be a file on disk or a message in another protocol. Unless used inside a secure protocol, artifacts must be signed and need to be provided along with an Owner Certificate and an Ownership Voucher (see terms), so the a device can validate the artifact’s signature to its Rightful Owner (see term).

Bootstrap Server: The term "bootstrap server" is used within this document to mean any RESTCONF server implementing the YANG module defined in Section 6.4.

Device: The term "device" is used throughout this document to refer to the network element that needs to be bootstrapped. The device is the RESTCONF client to a Bootstrap Server (see above) and, at the end of bootstrapping process, the device is the NETCONF or RESTCONF server to a deployment-specific NMS. See Section 5 for more information about devices.

Network Management System (NMS): The acronym "NMS" is used throughout this document to refer to the deployment specific management system that the bootstrapping process ultimately connects the devices to. From a device’s perspective, when the bootstrapping process has completed, the NMS is a NETCONF or RESTCONF client.
Owner: See Rightful Owner.

Owner Certificate: An owner certificate, signed by the device’s manufacturer or delegate, binds an owner identity to the owner’s private key, which the owner can subsequently use to sign artifacts. The owner certificate is an X.509 certificate encoding the owner’s identity in the Subject field of the X.509 certificate. The owner certificate is used by devices only when validating owner signatures on Signed Data (see term).

Ownership Voucher: An ownership voucher, signed by the device’s manufacturer or delegate, binds an owner identity to one or more device identities (e.g., serial numbers). The ownership voucher is used by devices only when validating owner signatures on Signed Data (see term).

Redirect Server: The term "redirect server" is used to refer to a Bootstrap Server (see above) that only returns Redirect Information (Section 2.4).

Rightful Owner: The rightful owner of a device is the person or organization that purchased the device. How ownership can be conveyed to a device is described in Section 2.3.

Secure Redirect: Secure redirect is like an HTTP Redirect except that it also returns TLS certificates that can be used as trust anchors to validate the secure connection to the Bootstrap Server the device is being redirected to.

Signed Data: The term "signed data" is used throughout to mean data that has been signed by a device’s Rightful Owner’s private key. Any time data is signed, it must be presented along with an Owner Certificate and Ownership Voucher (see terms).

Unsigned Data: The term "unsigned data" is used throughout to mean data that has not been signed by a device’s Rightful Owner’s private key. The option to use unsigned data is available only when the data is obtained over a secure connection, such as to a Redirect Server or a Bootstrap Server (see terms).

1.3. Tree Diagrams

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

- Brackets "[" and "]" enclose list keys.
o Braces "{" and "}" enclose feature names, and indicate that the named feature must be present for the subtree to be present.

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

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

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

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

2. Guiding Principles

This section provides overarching principles guiding the solution presented in this document.

2.1. Trust Anchors

A device in its factory default state can only trust remote keys for which it has preconfigured trust anchors. For instance, the device may have a trust anchor (e.g., a X.509 certificate) for when authenticating a very specific HTTPS server, and another trust anchor for when validating boot-image files, and yet another trust anchor for when verifying software licenses.

2.2. Conveying Trust

Trust can be conveyed by either transport level security or artifact signing. For instance, if a device connects to an HTTPS server, authenticating the TLS certificate to a known trust anchor, then any data the device receives from the HTTPS server can also be trusted. Likewise, if a device can authenticate the signature over some data to a known trust anchor, then that data can also be trusted. In general, any data obtained from a trusted source MAY be trusted and, any data obtained from an untrusted source MUST NOT be trusted.

It is possible but unnecessary to provide signed data over a secure connection. For instance, a device connecting to a trusted HTTPS server may retrieve data that has been signed by its rightful owner, but this is not required, as the device is already assured by the server that its data was staged by its rightful owner. That said, when an insecure connection is used (e.g., DHCP), the device has no choice but to require that the data be signed, in order to trust the data.
2.3. Ownership

The goal of this document is to enable a device to connect with its rightful owner’s NMS. This entails the manufacturer being able to track who owns which devices (out of the scope of this document), as well as an ability to convey that information to devices (in scope). Matching the two ways to convey trust, this document provides both a protocol-oriented solution as well as an artifact based solution for conveying ownership.

The protocol based solution conveys ownership by API contract, in that the server asserts that it will only return data that it is sure was staged by that device’s rightful owner. How ownership for a device is assured is out of scope of this document.

The artifact based solution involves the manufacturer signing an owner key and then later, when the ownership for devices is established, the manufacturer signing a voucher that assigns those devices to the owner, and then the owner using their private key to sign the artifacts. Thus, from the device’s perspective, it can use the presented "ownership voucher" to validate the presented "owner certificate", which it can then use to validate the signature over the presented artifact.

The YANG module in Section 6.4 includes grouping statements defining the format for the owner certificates and ownership vouchers used by the bootstrapping solution presented in this document.

2.4. Information Types

This document presumes there exists two types of zero touch information: redirect information and bootstrapping information. Either type of data may by accessed as unsigned data over a secure connection to a trusted server (e.g., HTTPS), or as signed artifacts obtained via an insecure method (DHCP server, removable storage device, etc.).

The redirect information type of data provides two bits of information: bootstrap server locations and trust anchors. The trust anchors are provided to enable the device to authentic the specified bootstrap servers (TLS certificate-based authentication). This is what distinguishes this technique from a standard HTTP Redirect and why it may sometimes be called "secure redirect".

The bootstrap information type of data provides information describing the boot-image and configuration the device should be running, in order to be considered bootstrapped. The boot-image
information is optional but, if it is provided, the device should install the boot image prior to installing the configuration.

The YANG module in Section 6.4 includes grouping statements defining the format for redirect and bootstrap information types used by the bootstrapping solution presented in this document.

3. Sources for Bootstrapping Data

Following are the sources of bootstrapping data that are referenced by the workflow presented in Section 4.3. Other sources for bootstrapping information may be described in other documents, so long as the principles for when the bootstrapping data needs to be signed or not are enforced.

Each of the descriptions below show how the bootstrapping data needs to be handled in a manner consistent with the guiding principles in Section 2.

For devices supporting more than one source for bootstrapping data, no particular sequencing order has to be observed, as each source is equally secure, in that the chain of trust always goes back to the same root of trust, the manufacturer.

3.1. Removable Storage

A device may attempt to read bootstrapping information from a directly attached removable storage device. This information would most likely have to be signed, as removable storage devices are generally not trustworthy.

The information loaded from a removable storage device may redirect the device to a bootstrap server (i.e., redirect information) or it may provide the boot image and configuration (i.e., bootstrapping information) directly. For when providing the information directly, even the raw boot image file could be on the removable storage device, making it a fully self-standing solution.

3.2. DHCP Server

A device may attempt to read bootstrapping information from a DHCP server (e.g., DHCP options). This information would have to be signed, as the DHCP protocol is not a secure protocol.

The information may again be either redirect or bootstrapping information. If bootstrapping information is provided, the URI to the boot image would have to specify a file server (e.g., ftp://, tftp://, etc.), as DHCP servers do not themselves distribute files.
Note that it is acceptable for boot images to be fetched using an insecure protocol when having an embedded signature, as is commonly the case.

3.3. Internet Based Service

A device may attempt to read bootstrapping information from a trusted Internet based service. The hosted information would not have to be signed, as the device would authenticate the service when establishing a secure connection to it, using trust anchors the device is manufactured with in its factory default state.

This document defines a RESTCONF API for a bootstrap server that may be hosted on the Internet. The YANG module describing this API is provided in Section 6.4.

The information may again redirect the device to a bootstrap server (i.e., redirect information) or it may direct the device to load a boot image and a configuration (i.e., bootstrapping information). If bootstrapping information is provided, the URI to the boot image would not have to be to a server the device has a trust anchor for, assuming the boot image has an embedded signature, as is commonly the case.

4. Workflow Overview

The zero touch solution presented in this document is conceptualized to be composed of the workflows described in this section. Implementations MAY vary in details.

4.1. Onboarding and Ordering Devices

The following diagram illustrates key interactions that occur from when a manufacturer or delegate onboards a prospective device owner to when the manufacturer ships devices for an order placed by the prospective device owner.
The interactions in the above diagram are described below.

1. A prospective owner establishes a trust relationship with a manufacturer in order to place zero touch orders. Assuming the manufacturer or delegate hosts a secure redirect server, this onboarding interaction might entail the creation of an online account that the owner can use to configure redirect information for future device orders. Alternatively, the onboarding interaction may include the manufacturer signing an owner certificate (see Section 1.2), to be used for bootstrapping devices not using the manufacturer’s redirect server. The onboarding interaction may also do both, giving the choice to the owner for how specific devices should bootstrap.
2. The prospective owner downloads from the manufacturer the X.509 based trust anchor certificate that can be used to validate the IDevID certificate [Std-802.1AR-2009] the devices will present as their SSH host key or TLS server certificate, when establishing a NETCONF or RESTCONF connection with the prospective owner’s deployment-specific NMS.

3. Some time later, the prospective owner places an order with the manufacturer, perhaps with a special flag checked for zero touch handling. At this time, perhaps before placing the order, the owner may model the devices in their NMS. That is, create virtual objects for the devices with no real-world device associations. For instance the model can be used to simulate the device’s location in the network and the configuration it should have when fully operational.

4. When the manufacturer ships the devices for the order, the manufacturer notifies the owner of the devices’ unique identifiers and shipping destinations, which the owner can use to stage the network for when the devices powers on. Additionally, the manufacturer may send a ownership voucher assigning ownership of those devices to the rightful owner and/or configure backend systems so the secure redirect service can associate the redirect information to the devices. The owner sets this information on the NMS, perhaps binding specific device identifiers and ownership vouchers (if supported) to specific modeled devices.

4.2. Owner Stages the Network for Bootstrap

The following diagram illustrates how an owner stages the network for bootstrapping devices.
The interactions in the above diagram are described below.

1. Having previously modeled the devices, including setting their fully operational configurations, associating device identifiers and ownership vouchers (if supported), the owner may "activate" one or more modeled devices. That is, tell the NMS to perform the steps necessary to prepare for when the real-world devices are powered up.

2. One thing the NMS must do is configure the deployment specific bootstrap server. Illustrated here as an external component, the bootstrap server may be implemented as an internal component of the NMS itself. Configuring the bootstrap server may occur via a programmatic API not defined by this document. This step sets signed or unsigned bootstrap information, as shown in Section 6.2, for the devices being activated. The configuration set MUST be at least enough to enable a secure NETCONF or
RESTCONF connection to be established and MAY be the device’s fully operational configuration.

3. If it is desired to use a manufacturer or delegate hosted redirect service to supply the bootstrapping information, the redirect server would need to be configured to supply the redirect information to the devices. Configuring the redirect server may occur via a programmatic API not defined by this document. This step sets signed or unsigned redirect information, as shown in Section 6.2, for the devices being activated. The redirect information MUST set the IP address or hostname of the deployment specific bootstrap server and MAY set the X.509 trust anchor certificate to authenticate the bootstrap server’s TLS certificate.

4. If it is desired to use a DHCP server to supply bootstrapping information, the DHCP server would need to be configured to supply the redirect information to the devices. Configuring the DHCP server may occur via a programmatic API (not defined by this document). Since DHCP is an insecure protocol, the information would have to be signed. That is, either signed redirect or signed bootstrap information, as shown in Section 6.2.

5. If it is desired to use a removable storage device (e.g., USB flash drive) to supply bootstrapping information, the information would need to be placed onto it. Since a removable storage device is insecure, the information would have to be signed. That is, either signed redirect or signed bootstrap information, as shown in Section 6.2.

4.3. Device Powers On

The following diagram illustrates how a device might behave when powered on. Note that this is merely exemplary, subject to which bootstrapping strategies the device supports, which may be more or less than depicted below.

This example sequences the sources of information (see Section 3) based on locality, or how "close" to the device the data is. Whether this sequence makes sense for a specific type of device needs to be determined by the manufacturer.
1. if not factory default, then exit.

2. check
   # if signed redirect information found
   #-----> # hook
   # either NMS-initiated connection
   #----->
   # or device-initiated connection
   #----->
   # else if signed bootstrap information found (call home)
   #----->

3. Get IP assignment
   #----->
   # if signed redirect information found
   #-----> # hook
   # either NMS-initiated connection
   #----->
   # or device-initiated connection
   #----->

4. check
   # if signed or unsigned redirect information found
   #-----> # hook
   # either NMS-initiated connection
   #----->
   # or device-initiated connection
   #----->

5. loop or wait for manual provisioning.
The interactions in the above diagram are described below.

1. Upon power being applied, the device’s bootstrapping logic first checks to see if it is running in its factory default state. If it has a modified state, then the bootstrapping logic would exit and none to the following interactions would occur.

2. If the device is able to load bootstrapping data from a removable storage device (e.g., USB flash drive), it might choose to do so first. The removable storage may have either signed redirect information or signed bootstrap information, as shown in Section 6.3.

   In the case that signed redirect information is found, the device would use it to established a connection to the deployment-specific bootstrap server, which would set its boot image and configure it to enable connections with the deployment-specific NMS to be established. If the bootstrap supports notifying (e.g., via a web-hook) external systems when a device sends its bootstrap-complete notification (Section 6.4), it would be possible for the NMS to initiate a NETCONF or RESTCONF connection to the device. Otherwise the configuration could configure the device it initiate a NETCONF or RESTCONF call home [draft-ietf-netconf-call-home] connection to the deployment-specific NMS.

   In the case that signed bootstrap information is found, the device would use it to set its boot image and initial configuration, which would have to direct it to initiate a NETCONF or RESTCONF call home connection to the deployment-specific NMS.

   If the device is unable to bootstrap using any of the information on the removable storage device, it would proceed to the next source of bootstrapping information, if any.

3. If the device is able to load bootstrapping data from a DHCP server, when obtaining a DHCP assignment, it may receive signed redirect information in a DHCP Option (Section 8). The device would process the signed redirect information in the same manner as described above for when it’s loaded from a removable storage device. If the device is unable to bootstrap using information provided by a DHCP server, it would proceed to the next source of bootstrapping information, if any.

4. If the device is able to obtain a routable address to the Internet, it may attempt to establish a connection to a redirect server that is set by its factory default state (Section 5.1).
These connections would use the RESTCONF API described in this document and would be secured using trust anchors also set in the device’s factory default state. The redirect server may provide signed or unsigned redirect information. In either case, the device would process the redirect information in the same manner as described above for when it’s loaded from a removable storage device. If the device is unable to bootstrap using information provided by any redirect servers, it would proceed to the next source of bootstrapping information, if any.

5. If no more sources of bootstrapping information are available, the device may fall into a loop to try again or it may provide manageability interfaces for manual configuration (e.g., CLI, HTTP, NETCONF, etc.).

5. Device Details

Devices supporting Zero Touch MUST have the preconfigured factory default state and bootstrapping logic described in the following sections.

5.1. Factory Default State

```
<device>
  <read-only storage>
    1. list of public Internet Bootstrap Servers
    2. list of trust anchor certs for Bootstrap Servers
    3. trust anchor cert for owner certificates
    4. trust anchor cert for device ownership vouchers
    5. IDevID cert & associated intermediate certificate(s)
  <secure storage>
    6. private key
```

1. Devices that support loading bootstrapping information from the Internet (see Section 3) MUST be manufactured with a list of default Bootstrap Servers. Each Bootstrap Server may be identified via a hostname or an IP address.

2. Devices that support loading bootstrapping information from the Internet (see Section 3) SHOULD be manufactured with a list of trust anchor certificates that can be used to authenticate the Bootstrap Server connections with.

3. Devices that support loading owner signed data (see Section 1.2) MUST be manufactured with the trust anchor certificate for the Owner Certificates that the Manufacturer provides to prospective owners when they enroll in the Manufacturer’s Zero Touch program (see Section 4.1).

4. Devices that support loading owner signed data (see Section 1.2) MUST also be manufactured with the trust anchor certificate for the device Ownership Vouchers that the Manufacturer provides to prospective owners when it ships out an order of Zero Touch devices (see Section 4.1).

5. Devices MUST be manufactured with an initial device identifier (IDevID), as defined in [Std-802.1AR-2009]. The IDevID is an X.509 certificate, encoding a globally unique device identifier (e.g., serial number). The device MUST also possess any intermediate certificates between the IDevID certificate and the Manufacturer’s IDevID trust anchor certificate.

6. Device MUST be manufactured with a private key that corresponds to the public key encoded in the device’s IDevID certificate. This private key SHOULD be securely stored, ideally by a cryptographic processor (e.g., a TPM).

5.2. Boot Sequence

A device claiming to support Zero Touch MUST support the boot sequence described in this section.
Power On

1. Running default config?  --------> Boot normally
   Yes

2. For each supported source for bootstrapping information, try to load bootstrapping data from the source
   Yes

3. Able to bootstrap off any source?  -----> Run with new configuration
   No

4. Loop or wait for manual provisioning.

These interactions are described next.

1. When the device powers on, it first checks to see if it is running the factory default configuration. If it is running a modified configuration, then it boots normally.

2. The device iterates over its list of sources for bootstrapping information. Details for handling different types of sources are provided in subsequent sections.

3. If the device is able to bootstrap itself off any of the sources for bootstrapping information, it runs with the new bootstrapped configuration merged into its running datastore.

4. Otherwise the device MAY loop back through the list of bootstrapping sources again or wait for manual provisioning.

When the source is a removable storage device, the device MUST be able to read from it signed data (see term) and validate that the data was signed by its rightful owner, using the algorithm in Section 5.3.

When the source is a DHCP server, the device MUST be able to read from it signed data (see term) and validate that the data was signed by its rightful owner, using the algorithm in Section 5.3.

When the source is a bootstrap server, that is, using the RESTCONF API presented in Section 6.4, the device MUST be able to authenticate the server using one of the the device’s preconfigured trust anchors.
Once done authenticating the bootstrap server, the device MUST attempt to fetch the bootstrapping data hosted for it there, using its unique identifier (e.g., serial number) as the key into the "device" list. If bootstrapping data is found and it is signed, then the device MUST first validate that the data was signed by its rightful owner using the algorithm in Section 5.3. The device then processes the bootstrapping data as described in Section 5.4. The device MAY post progress notification messages to the server, but SHOULD only do so if it has first authenticated itself to the server (e.g., client authentication).

5.3. Validating Signed Data

If the device is ever presented signed data, it MUST validate the signed data as described in this section.

Whenever there is signed data, the device MUST also be provided an Ownership Voucher and an Owner Certificate.

The device MUST first authenticate the Ownership Voucher by validating the signature on it to one of its preconfigured trust anchors (see Section 5.1) and verify that the voucher contains the device’s unique identifier (e.g., serial number). If the authentication of the voucher is successful, the device extracts the Rightful Owner’s identity from the voucher for use in the next step.

Next the device MUST authenticate the Owner Certificate by performing X.509 certificate path validation on it to one of its preconfigured trust anchors (see Section 5.1) and by verifying that the Subject contained in the certificate matches the Rightful Owner identity extracted from the voucher in the previous step. If the authentication of the certificate is successful, the device extracts the Owner’s public key from the certificate for use in the next step.

Finally the device MUST authenticate the signed data by verifying the signature on it was generated by the private key matching the public key extracted from the Owner Certificate in the previous step.

If any of these steps fail, then the device MUST mark the data as invalid and not perform any of the subsequent steps.

5.4. Processing Bootstrap Data

In order to process bootstrapping data, the device MUST follow the steps presented in this section.

If the data is redirect-information (see Section 2.4), the device MUST immediately attempt to establish a RESTCONF connection to the
provided bootstrap server IP address or hostname. If a hostname is provided and DNS resolves it to more than one IP address, the device MUST attempt to try to connect to all of them, until it is able to successfully bootstrap off one of them. The device MUST authenticate the bootstrap server’s TLS certificate using the X.509 certificate provided by the redirect information.

If the data is bootstrap-information (see Section 2.4), the device MUST first check if it contains any boot-image information and, if so, check to see if it differs from what the device is currently running and, if so, install the boot-image using the provided URI and reboot (Note, it is assumed that the boot-image contains an embedded signature that the installation step will verify). This will cause the device’s bootstrap logic to restart, which will again come to this point, though with a matching boot-image, thus letting the device to proceed past this step. Next the device MUST process the configuration contained in the bootstrapping information, by merging it into its running configuration.

At this point, the device has completely processed the bootstrapping data and is "bootstrap complete". If the configuration configured the device it initiate a call home connection, it would proceed to do so now. Otherwise, the device would wait for a NETCONF or RESTCONF client to connect to it.

6. YANG-defined API and Artifacts

Central to the solution presented in this document is the use of a YANG module [RFC6020] to simultaneously define a RESTCONF based API for a bootstrap/redirect server as well as the encoding for signed artifacts that can be conveyed outside of the RESTCONF protocol (DHCP, FTP, TFTP, etc.).

6.1. Module Overview

The following tree diagram Section 1.3 provides an overview for both the API and artifacts that can be used outside of RESTCONF.
module: ietf-zerotouch-bootstrap-server
  +--ro devices
  |    +--ro device* [unique-id]
  |    |    +--ro unique-id string
  |    +--ro (type)?
  |    |    +--:(redirect-information)
  |    |    |    +--ro redirect-information
  |    |    |    |    +--ro address inet:host
  |    |    |    |    +--ro trust-anchor binary
  |    |    |    |    +--ro signature? string
  |    |    +--:(bootstrap-information)
  |    |    |    +--ro bootstrap-information
  |    |    |    |    +--ro boot-image
  |    |    |    |    |    +--ro name string
  |    |    |    |    |    +--ro md5 string
  |    |    |    |    |    +--ro sha1 string
  |    |    |    |    |    +--ro path string
  |    |    |    |    |    +--ro signature? string
  |    |    +--ro configuration
  |    |    |    +--ro config
  |    |    |    |    +--ro signature? string
  |    +--ro ownership-voucher
  |    |    +--ro voucher binary
  |    |    +--ro issuer-crl? string
  |    +--ro owner-certificate
  |    |    +--ro certificate string
  |    |    +--ro issuer-crl? string
  +---x notification
  +---w input
  |    +---w type enumeration
  |    +---w message? string
  |    +---w ssh-host-keys
  |    |    +---w ssh-host-key* enumeration
  |    |    +---w format string
  |    |    +---w key-data string

In the above diagram, notice that all of the protocol accessible node
are read-only, to assert that devices can only pull data from the
bootstrap server.

Also notice that the module defines an action statement, which
devices may use to provide progress notifications to the Bootstrap
Server.
6.2. API Examples

This section presents some examples illustrating device interactions with a Bootstrap Server to access Redirect and Bootstrap information, both unsigned and signed, as well as to send a progress notification.

6.2.1. Unsigned Redirect Information

The following example illustrates a device using the API to fetch its bootstrapping data. In this example, the device receives unsigned redirect information. This example is representative of a response a well-known Internet facing redirect service might return.
6.2.2. Signed Redirect Information

The following example illustrates a device using the API to fetch its bootstrapping data. In this example, the device receives signed redirect information. This example is representative of a response that redirect service might return if concerned the device might not be able to authenticate its TLS certificate.

REQUEST
-------

GET https://example.com/restconf/data/ietf-zerotouch-bootstrap-server::devices/device=123456 HTTP/1.1
HOST: example.com
Accept: application/yang.data+xml

RESPONSE
--------

HTTP/1.1 200 OK
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+xml

  <device>
    <unique-id>123456789</unique-id>
    <redirect-information>
      <address>phs.example.com</address>
      <trust-anchor>
        WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM11QlIsdWpOcjFTMmRLR05EMUc20VJpK2FVNGw2NTd2NCTadVJMgprYjzkS4Fw4SDwvXBCYnA4dmtNanFtZjJma3Rq2HZxExFppUUtTbndWZTF2ZwotNGcEk3UE90cnNFvIjRWUNBd0VBQWFPQ0FSSXdnZ0VFck1CMBdVMVkrRgdVEJIz0JTWe4dbUEKMNhpRHR0TVkvvHFLNw5d4cFJBJz1OYU0cEZRzd05ERV6QVJCZ05WQkFNVerT1NU0Q0KYzNOMvpYS0NDUUNVRHSN16UG8zREFNQmdOVkhSTUJBZjhFCKFqQUFQTRHQTFVZER3RUIvd1FFQxdJSGdEQpBCCZ05WSF14RVlqMndNRj2nxSFBrZ2hnW9kSFJ3T2k4dlpYaGkYkYHCC1pTNWpiMjB2WlhooaGJQtQnNaUzVqY215a9LUTJNRF4Fq3pBskJnT12CQV10QWQmdOVkJBwVrBBfZ2UTVJBd0tNwUWUFLX2dbApR0ZOY0d4eB1RNdEQQMkF6a3hqUD1VQWtHR0dvS1UeUc1SVROWm0vK3B0R2FieXVDMjB3Rd2kv2Z25PZnpZNEhONApXYopTaUp2K2xtYW3s3RTRORUZXZ9Rdp4NU1XZmdvN2RJSU5QFR5tS0Cgg=
      </trust-anchor>
    </redirect-information>
  </device>
</devices>
GET https://example.com/restconf/data/ietf-zerotouch-bootstrap-server::devices/device=123456 HTTP/1.1
HOST: example.com
Accept: application/yang.data+xml

RESPONSE
--------
HTTP/1.1 200 OK
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+xml

  <device>
    <unique-id>123456789</unique-id>
    <redirect-information>
      <address>phs.example.com</address>
      <trust-anchor>
        WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc30EgrRkYyRTFwdSt42ZVRJbVVFm
        l1Q11sdWpoCjFTMnLR05EMUc20VJpK2FWNGw2N7d2N7cadvJVMzgprYjk
        zSNFwSDwVXBCYnA4dmtNanFtZjJma3Rq2HBxeFppUUtBndWZTF2Zwot
        NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdn2OVPCk1CMEdBMVvkRgD
        VEJ1Z0JTWe61bUEKmhpRHV0TVkvVHFLNWd4cFJBZ1Z0UY0cERd2z05ER
        V6OvJcZ05QwpkFNVENrT1NUQ0JKyZiNOMVypY0NDUNVHNHS116UG8zRREF
        NQmdOvkhsSTUJBzjhcFckFgQUNFQTRHTFV2ZER3U1vd1FFQXjdJSgdEenBc
        Z05WSF14RvlqMrdNjznSXFBZ2hoNW9ksFJ3T2k4dlpYaGyKy1cC1pTN
        WpiJNjgb2WlhoaGJYQnaUzVqY215aU9LUTJNRF4Q3pBSkJnTlZCQV1UQW
        QmdOVkJBWVBfBZUTVJbD0rnWURWUVFLRXdkbAp1R0ZY0d4b1E1RNdHdQ
        MxK6a3hqUD1WQTdHR0dvS1UeUC1SR0Wm0vK3B0R2FieXVDMyBRd2kvZ
        25P2nNZHnONaPyYpTaUp2K2xtYW3z3RTORUXZS9Gp4NU1XZmdvN2
        RJSUJQFRSTs5OCg==
      </trust-anchor>
    </redirect-information>
    <ownership-voucher>
      <voucher>
        ChQQSnVuaXBlc19O2XR3b3JrczEdMBsGA1UECxDQQ2VydGlmaWNhdGVfSXNzdWFu
        Y2UxGTAXBgNVBAUEFQRQTV9UcnVzdF9BbmNob3IxHTAbBgkqhkiG9w0BCQEQDwY
        MBEGAlUEchQKBVFBNX12lbmRvcjEJeMBcGA1UEAxQQSnVuaXBlc19YWFhYWF9DQTCC
        AS1wDQYJKoZIhvcNAAEQBBQADgEFADCCAQgEBANL5Mk5qEsVuqO+JmXWLMxFi
        yh/JafTW7ifm3K8zdQ2MIIHFBgNVHSMeGdcwSASdfsJzNnmTN5b+CDuJGLyDal
        WFPaoYGwpIGtMMGcMQswCQYDVQQGEwJvUzEMBEGAlUECBEKMQ2FpsaWZvcm5pYTES
        MBAGAlUEBkzMU3ubn1ZWx1MRk6wYDVQQFBBKdN5pcGvyX051dhcvcmztZMr0W
        GwYDVQQFLBRZJX0aWZpY2F0ZV9jM3N1YlW5jZTEZMBcGA1UEAxQQVFBNX1RydXN0
        X0FU2hvcjEdMBsGCSqGSIb3DQEJARY0F2AanVuaXBlc15b22CCQUfUsEdTn5v
      </voucher>
  </device>
</devices>
6.2.3. Unsigned Bootstrap Information

The following example illustrates a device using the API to fetch its bootstrapping data. In this example, the device receives unsigned bootstrapping information. This example is representative of a response a locally deployed bootstrap server might return.

REQUEST
-------
GET https://example.com/restconf/data/ietf-zerotouch-bootstrap-server::devices/device=123456 HTTP/1.1
HOST: example.com
Accept: application/yang.data+xml

RESPONSE
--------
HTTP/1.1 200 OK
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+xml

  <device>
    <unique-id>123456789</unique-id>
    <bootstrap-information>
      <boot-image>
        <name>boot-image-v3.2R1.6.img</name>
        <md5>SomeMD5String</md5>
        <sha1>SomeSha1String</sha1>
        <path>/some/path/to/raw/file</path>
      </boot-image>
    </bootstrap-information>
  </device>
</devices>
<configuration>
  <config>
    <!-- from ietf-system.yang -->
    <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
      <authentication>
        <user>
          <name>admin</name>
          <ssh-key>
            <name>admin's rsa ssh host-key</name>
            <algorithm>ssh-rsa</algorithm>
            <key-data>AAAAB3NzaC1yc2EAAAADAQABAAABAQDeJMV8zrtsi8CgsESRCjczfve2m6zD3awSBPrh7ICgQLQvHVbPL89eH1uecStKL3HrEgXaI/O2MwjE1L9YxLze5p2ngzK61vikUSqfMukeBohFTRDZ8bUtrF+HMLlTrnoCVCcWAw1lo99DGDAuw66G45gLHbHgqMtqXkZdxU9kkx/fL325G76Fy6sA5vg7SLqQPjXKft2CAhin8xwYR2y6r/2N9PMJ2Dnepvq4H2DKqB1e340jWqEIuA7LvEYql4unq4Iog+/+CiumTkmmQINRgIoj4FCzYk09NvRE6f0SLf6gakWVO2ZqQ8929uWjCW1Glqn2mPibp2Go1</key-data>
          </ssh-key>
        </user>
      </authentication>
    </system>
    <!-- from ietf-netconf-server.yang -->
    <netconf-server xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
      <call-home>
        <application>
          <name>config-mgr</name>
          <ssh>
            <endpoints>
              <endpoint>
                <name>east-data-center</name>
                <address>11.22.33.44</address>
              </endpoint>
              <endpoint>
                <name>west-data-center</name>
                <address>55.66.77.88</address>
              </endpoint>
            </endpoints>
            <host-keys>
              <host-key>my-call-home-x509-key</host-key>
            </host-keys>
          </ssh>
        </application>
      </call-home>
    </netconf-server>
  </config>
</configuration>
6.2.4. Signed Bootstrap Information

The following example illustrates a device using the API to fetch its bootstrapping data. In this example, the device receives signed bootstrapping information. This example is representative of a response that bootstrap service might return if concerned the device might not be able to authenticate its TLS certificate.

REQUEST
-------
GET https://example.com/restconf/data/ietf-zerotouch-bootstrap-server::devices/device=123456 HTTP/1.1
HOST: example.com
Accept: application/yang.data+xml

RESPONSE
--------
HTTP/1.1 200 OK
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+xml

  <device>
    <unique-id>123456789</unique-id>
    <bootstrap-information>
      <boot-image>
        <name>boot-image-v3.2R1.6.img</name>
        <md5>SomeMD5String</md5>
        <sha1>SomeSha1String</sha1>
        <path>/some/path/to/raw/file</path>
        <signature>SomeSignatureString</signature>
      </boot-image>
    </bootstrap-information>
  </device>
</devices>
<config>
    <!-- from ietf-system.yang -->
    <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
        <authentication>
            <user>
                <name>admin</name>
                <ssh-key>
                    <name>admin’s rsa ssh host-key</name>
                    <algorithm>ssh-rsa</algorithm>
                    <key-data>AAAAB3NzaC1yc2EAAAADAQABAAABAQDeJMV8zrtsi8CgEsRCjCzvfeZm6zD3awSBPvh7IcgLQvHvbpl89eHluecStKl3HrEqXaI/O2MwjE11G9YxLze55p2ngzK6vikkUSqFMukeB0hFTrDZ8bUtrF+HML1TRnoCVCcWAw110r91DGDAuw6G45LchAlHamBtQxKndzU99j/fLz3S5G76Fy6sA5vg7SlqFPFjXXft2CAhin8xwYRzy6r/2N9PMJ2Dnepvq4H2DKqBle340jWqEIuAT7lVeJylq4unq4Iog/+<Ciim7kmqIMRgi0j4FCzYk9NvRE6fOSLLf6gakWVOZzq8929uwJrWjGlnqn2mPibp2Go1</key-data>
                </ssh-key>
            </user>
        </authentication>
    </system>
</config>

<!-- from ietf-netconf-server.yang -->
<netconf-server xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
    <call-home>
        <application>
            <name>config-mgr</name>
            <ssh>
                <endpoints>
                    <endpoint>
                        <name>east-data-center</name>
                        <address>11.22.33.44</address>
                    </endpoint>
                    <endpoint>
                        <name>west-data-center</name>
                        <address>55.66.77.88</address>
                    </endpoint>
                </endpoints>
                <host-keys>
                    <host-key>my-call-home-x509-key</host-key>
                </host-keys>
            </ssh>
        </application>
    </call-home>
</netconf-server>
6.2.5.  Progress Notifications

The following example illustrates a device using the API to post a notification to the server. The device may send more than one notification to the server (e.g., to provide status updates). The YANG module defines only one notification type, bootstrap-complete. Other notification types may be defined through YANG augmentation.

The bootstrap server MUST NOT process a notification from a device without first authenticating the device. This is in contrast to when a device is fetching data from the server, a read-only operation, in which case device authentication is not strictly required.

In this example, the device sends a notification indicating that it has completed bootstrapping off the data provided by the server. This example also illustrates the device sending its SSH host keys to the bootstrap server, which it might, for example, forward onto a downstream NMS component, so that it can subsequently authenticate the device when establishing a NETCONF over SSH connection to it.

A device providing its SSH host key or TLS server certificate is not needed when the device has an IDevID certificate [Std-802.1AR-2009] and is able to present the IDevID certificate as its SSH host key or TLS server certificate, when establishing a NETCONF or RESTCONF connection.
REQUEST
-------
POST https://example.com/restconf/data/ietf-zerotouch-bootstrap-server::devices/
device=123456/notification HTTP/1.1
HOST: example.com
Content-Type: application/yang.data+xml

<input xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server">
  <notification-type>bootstrap-complete</notification-type>
  <message>example message</message>
  <ssh-host-keys>
    <ssh-host-key>
      <format>ssh-rsa</format>
      <key-data>AAAAB3NzaC1yc2EAAAADAQABAAABAQDeJMV8zrtsi8CgEsRCjCzfve2m6zD3awSBPrh7ICggLQvHVbPL89eHLuec5kL3HrEgXaI/O2MwjE1lG9YxlZeS5p2ngk61viKUSqFmukeBohFTrDZ8bUtrF+HMLITRncOCVcCWa11Or91DGDAuw6G45gLcHAlHmMtQkXnZdzU9kx/fL3ZS5G76Fy6sA5vg75LqQPJXXfC2AhIn8xwYRZy6r/2N9PMJ2Dnepvq4H2DQbIe340jWqELvEJyql4unq4Iog/+CiiumTkq1WRgIo4FCzYk09NE6fOSLLf6gakWVOZZgQ8929uWjCW1Glqnm2Pibp2Go1</key-data>
    </ssh-host-key>
    <ssh-host-key>
      <format>ssh-dsa</format>
      <key-data>AAAAB3NzaC1yc2EAAAADAQABAAABAQDeJMV8zrtsi8CgEsRCjCzfve2m6zD3awSBPrh7ICggLQvHVbPL89eHLuec5kL3HrEgXaI/O2MwjE1lG9YxlZeS5p2ngk61viKUSqFmukeBohFTrDZ8bUtrF+HMLITRncOCVcCWa11Or91DGDAuw6G45gLcHAlHmMtQkXnZdzU9kx/fL3ZS5G76Fy6sA5vg75LqQPJXXfC2AhIn8xwYRZy6r/2N9PMJ2Dnepvq4H2DQbIe340jWqELvEJyql4unq4Iog/+CiiumTkq1WRgIo4FCzYk09NE6fOSLLf6gakWVOZZgQ8929uWjCW1Glqnm2Pibp2Go1</key-data>
    </ssh-host-key>
  </ssh-host-keys>
</input>

RESPONSE
--------
HTTP/1.1 204 No Content
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server

6.3. Artifact Examples

This section presents some examples for how the same information provided by the API can be packaged into stand alone artifacts. The encoding for these artifacts is the same as if an HTTP GET request had been sent to the RESTCONF URL for the specific resource.

Encoding these artifacts for use outside of the RESTCONF protocol extends their utility for other deployment scenarios, such as when a local DHCP server or a removable storage device is used. By way of example, this may be done to address an inability for the device to access an Internet facing bootstrap/redirect server, or just for a preference to use locally deployed infrastructure.

6.3.1. Signed Redirect Information

The following example illustrates how a redirect can be encoded into an artifact for use outside of the RESTCONF protocol. The redirect information is signed so that it is secure even when no transport-level security is provided.

6.3.2. Signed Bootstrap Information

The following example illustrates how bootstrapping data can be encoded into an artifact for use outside of the RESTCONF protocol. The bootstrapping information is signed so that it is secure when no transport-level security is provided.

```xml
<bootstrap-information xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server">
    <boot-image>
        <name>boot-image-v3.2R1.6.img</name>
        <md5>SomeMD5String</md5>
        <sha1>SomeSha1String</sha1>
        <path>/some/path/to/raw/file</path>
        <signature>SomeSignatureString</signature>
    </boot-image>
</bootstrap-information>
```
<config>
  <!-- from ietf-system.yang -->
  <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
    <authentication>
      <user>
        <name>admin</name>
        <ssh-key>
          <name>admin’s rsa ssh host-key</name>
          <algorithm>ssh-rsa</algorithm>
          <key-data>AAAAB3NzaC1yc2EAAAADAQABAAABAQDeJMV8zrtsi8CgEsRCjcZfve2m6zD3awSBPh71CggLQvHvbFL89eHLuecStKL3HrEgXaI/O2MwjEl1G9YxLze55p2ngzK61vikUSqfMukeBohFTD8bUtrF+HMLlTRnoCVCWAw1lOr91DGDAuw6G45gLcHalHMmBtdxKsdzU99x/fL3Z5G76Fy6sA5
vgs7LSqFFjXxt2CAhin8xwYRzy6r/2N9PmJ2Dnepvq4H2DKqBle340jWqEiuA7LvE3Jy14unq4Iog++/CiumTkmgIKW9gIoj4FCxYkO9NgRE6fQSLlf6gskhW0ZzgQ8929uWjCWlGln2zmPfibp2GoI</key-data>
        </ssh-key>
      </user>
    </authentication>
  </system>
  <!-- from ietf-netconf-server.yang -->
  <netconf-server xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
    <call-home>
      <application>
        <name>config-mgr</name>
        <ssh>
          <endpoints>
            <endpoint>
              <name>east-data-center</name>
              <address>11.22.33.44</address>
            </endpoint>
            <endpoint>
              <name>west-data-center</name>
              <address>55.66.77.88</address>
            </endpoint>
          </endpoints>
          <host-keys>
            <host-key>my-call-home-x509-key</host-key>
          </host-keys>
        </ssh>
      </application>
    </call-home>
  </netconf-server>
</config>
6.3.3. Owner Certificate

The following example illustrates how the owner certificate, along with its CRL, can be encoded into an artifact for use outside of the RESTCONF protocol. As the Owner Certificate and CRL are already signed by the manufacturer, an additional owner signature is unnecessary.
6.3.4. Ownership Voucher

The following example illustrates how the ownership voucher, along with its CRL, can be encoded into an artifact for use outside of the RESTCONF protocol. As the Ownership Voucher and CRL are already encoded into an artifact, they can be used as a means to verify the ownership of a network device.

signed by the manufacturer, an additional owner signature is unnecessary.

6.4.  YANG Module

The bootstrap server's device-facing interface is normatively defined by the following YANG module:

```yang
module ietf-zero-touch-bootstrap-server {

  prefix "ztbs";

  import ietf-inet-types { // RFC 6991
    prefix inet;
  }

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

  ...
This module defines the southbound interface for Zero Touch bootstrap servers.

Copyright (c) 2014 IETF Trust and the persons identified as authors of the code. All rights reserved.

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

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

revision "2015-10-19" {
    description
        "Initial version";
    reference
        "RFC XXXX: Zero Touch Provisioning for NETCONF Call Home";
}

grouping redirect-information-grouping {
    description
        "This container contains information the device may use to redirect it to another bootstrap server.";

    leaf address {
        type inet:host;
        mandatory true;
        description
            "The IP address or hostname of the bootstrap server the device should redirect to.";
    }
}
leaf trust-anchor {
  type binary;
  mandatory true;
  description
    "A certificate that a device can use as a trust anchor to
    authenticate the bootstrap server it is being redirected
to. The binary certificate structure as specified by RFC
5246, Section 7.4.6, i.e.,: opaque ASN.1Cert<1..2^24>;
    ";
  reference
    "RFC 5246: The Transport Layer Security (TLS)
    Protocol Version 1.2";
}

leaf signature {
  type string;
  must "./../ownership-voucher";
  description
    "The signature over the concatenation of the previous leafs
    using the organization’s private key. Specifically,
    sign(name+md5+sha1+path), where simple string concatenation
    to join values is used, resulting in a single null-terminated
    string.";
}

}
leaf sha1 {
    type string;
    mandatory true;
    description
        "The output of the SHA-1 hash algorithm over the image file."
}

leaf path {
    type string;
    mandatory true;
    description
        "An absolute path to the boot-image file hosted on this
        Bootstrap server."
}

leaf signature {
    type string;
    must ../../../ownership-voucher;
    description
        "The signature over the concatenation of the previous leaves
        using the organization’s private key. Specifically,
        sign(name+md5+sha1+path), where simple string concatenation
        to join values is used, resulting in a single null-terminated
        string."
}

carriage configuration {
    type string;
    mandatory true;
    description
        "It is intended that the device will fetch this container
        as a whole, as its contents need to be processed together."
    anyxml config {
        mandatory true;
        description
            "Any configuration data model known to the device. It may
            contain Vendor-specific and/or standards-based data models.
            An example configuration using a couple IETF-defined data
            models is presented the Appendix of RFC XXXX."
    }
    leaf signature {
        type string;
        must ../../../ownership-voucher;
        description
            "The signature over the concatenation of the previous leaf
            using the organization’s private key. Specifically,
            sign(config), where ’config’ is treated as a single null-
            terminated string."
    }
}
grouping owner-certificate-grouping {

  leaf certificate {
    type string;
    mandatory true;
    description
    "This is an X.509 certificate, signed by a Vendor, for
    a business organization. This certificate must encode a
    Vendor-assigned value identifying the organization. This
    identifier must match the owner identifier encoded in
    the Ownership Voucher.";
  }

  leaf issuer-crl {
    type string;
    description
    "An optional CRL for the issuer used by the
    Vendor to sign Owner Certificates. The CRL should be
    as up to date as possible. This leaf is optional as
    it is primarily to support deployments where the device
    is unable to download the CRL from the CRL distribution
    point URLs listed in the Vendor’s trust anchor
    certificate.";
  }
}

grouping ownership-voucher-grouping {

  leaf voucher {
    type binary;
    mandatory true;
    description
    "A Vendor-specific encoding binding unique device
    identifiers to an owner identifier value matching the
    value encoded in the owner-certificate below. An
    example format for a voucher is presented in the
    Appendix of RFC XXXX.";
  }

  leaf issuer-crl {
    type string;
    description
    "An optional CRL for the issuer used by the
    Vendor to sign Ownership Vouchers. The CRL should be
    as up to date as possible. This leaf is optional as
    it is primarily to support deployments where the device
    is unable to download the CRL from the CRL distribution
    point URLs listed in the Vendor’s trust anchor
    certificate.";
  }
}
container devices {
    config false;
    description
        "A read-only list of device entries";
    list device {
        key unique-id;
        leaf unique-id {
            type string;
            description
                "A unique identifier for the device (e.g., serial number).
                Each device accesses its bootstrapping record by its unique
                identifier.";
        }
    }

    choice type {
        container redirect-information {
            uses redirect-information-grouping;
        }
        container bootstrap-information {
            uses bootstrap-information-grouping;
        }
    }

    container ownership-voucher {
        description
            "This container contains the Ownership Voucher that the
device uses to ascertain the identity of its rightful
owner, as certified by its Vendor.";

        when "./redirect-information/signature or ..)/bootstrap-information/*/signature";
        //must "./owner-certificate and ./redirect-information/signature or ..
        /bootstrap-information/*/signature";
        must "./owner-certificate";

        uses ownership-voucher-grouping;
    }

    container owner-certificate {
        description
            "It is intended that the device will fetch this container
as a whole, as it contains values that need to be
processed together.

when ".../ownership-voucher"
//must ".../ownership-voucher and ../redirect-information/signature or ..
/bootstrap-information/*/signature"

uses owner-certificate-grouping;

}

action notification {
  input {
    leaf type {
      type enumeration {
        enum bootstrap-complete {
          description
"Indicates that the device successfully processed the
bootstrap data, that is currently running the specified
boot image and has committed the configuration. At
this point, the device is ready to be managed by an
external NMS system. The device is never expected
access the bootstrap server again, unless reset to
its factory default again.";
        }
      }
    }
    mandatory true;
  }
  leaf message {
    type string;
    description
"A human-readable value.";
  }
  container ssh-host-keys {
    list ssh-host-key {
      when ".../type = bootstrap-complete";
      leaf format {
        type enumeration {
          enum ssh-dss;
          enum ssh-rsa;
        }
        mandatory true;
      }
      leaf key-data {
        type string;
        mandatory true;
      }
    }
  }
}
7. Security Considerations

7.1. Immutable storage for trust anchors

Devices MUST ensure that all their trust anchor certificates, including those for the Owner Certificate and Ownership Voucher, are protected from external modification.

It may be necessary to update these certificates over time (e.g., the manufacturer wants to delegate trust to a new CA). It is therefore expected that devices MAY update these trust anchors when needed through a verifiable process, such as a software upgrade using signed software images.

7.2. Real time clock

The solution for signed data includes validating Owner Certificates and Ownership Vouchers, each of which may contain expirations. Further, the solution includes using a CRLs, which also require freshness. Device implementations should take care to ensure the devices have a reliable clock when processing signed data.

7.3. Entropy loss over time

Section 7.2.7.2 of the IEEE Std 802.1AR-2009 standard says that IDevID certificate should never expire (i.e. having a notAfter 99991231235959Z). Given the long-lived nature of these certificates, it is paramount to use a strong key length (e.g., 512-bit ECC). Manufacturers SHOULD deploy Online Certificate State Protocol (OCSP) responders or CRL Distribution Points (CDP) to revoke certificates in case necessary.

7.4. Serial Numbers

This draft suggests using the device’s serial number as the unique identifier in its IDevID certificate. This is because serial numbers are ubiquitous and prominently contained in invoices and on labels affixed to devices and their packaging. That said, serial numbers many times encode revealing information, such as the device’s model...
number, manufacture date, and/or sequence number. Knowledge of this information may provide an adversary with details needed to launch an attack.

8. IANA Considerations

Editor Note: this section needs to be rewritten to use the redirect and bootstrap information types (see Section 2.4).

8.1. Zero Touch Information DHCP Options

The following registrations are in accordance to RFC 2939 for "BOOTP Manufacturer Extensions and DHCP Options" registry maintained at http://www.iana.org/assignments/bootp-dhcp-parameters.

8.1.1. DHCP v4 Option

Tag: XXX

Name: Zero Touch Information

Description: Returns a list of null-terminated Configuration Server hostnames and/or IP addresses.

<table>
<thead>
<tr>
<th>Code</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXX</td>
<td>n</td>
</tr>
<tr>
<td>svr1</td>
<td>svr2</td>
</tr>
</tbody>
</table>

Reference: RFC XXXX

8.1.2. DHCP v6 Option

Tag: YYY

Name: Zero Touch Information

Description: Returns a list of null-terminated Configuration Server hostnames and/or IP addresses.

<table>
<thead>
<tr>
<th>Code</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYY</td>
<td>n</td>
</tr>
<tr>
<td>svr1</td>
<td>svr2</td>
</tr>
</tbody>
</table>

Reference: RFC XXXX
9. Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): David Harrington, Dean Bogdanovic, Martin Bjorklund, Max Pritikin, Stephen Hanna, Wes Hardaker, Russ Mundy, Reinaldo Penno, Randy Presuhn, Juergen Schoenwaelder.

Special thanks goes to Steve Hanna, Russ Mundy, and Wes Hardaker for brainstorming the original I-D’s solution during the IETF 87 meeting in Berlin.

10. References

10.1. Normative References


10.2. Informative References


Appendix A. Examples

A.1. Ownership Voucher

Following describes an example data-model for an Ownership Voucher. Real vouchers are expected to be encoded in a Manufacturer-specific format outside the of scope for this draft.

A tree diagram describing an Ownership Voucher:

```
module: ietf-zerotouch-ownership-voucher
  +--rw voucher
    +--rw owner-id     string
    +--rw unique-id*   string
    +--rw created-on   yang:date-and-time
    +--rw expires-on?  yang:date-and-time
    +--rw signature    string
```

The YANG module for this example voucher:

```
<CODE BEGINS> file "ietf-zerotouch-ownership-voucher@2015-10-19.yang"

module ietf-zerotouch-ownership-voucher {
  prefix "ztov";

  import ietf-yang-types { prefix yang; }

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

  contact "WG Web: <http://tools.ietf.org/wg/netconf/>
          WG List: <mailto:netconf@ietf.org>
          WG Chair: Mehmet Ersue
                     <mailto:mehmet.ersue@nsn.com>
          WG Chair: Mahesh Jethanandani
                     <mailto:mjethanandani@gmail.com>
          Editor: Kent Watsen
                     <mailto:kwatsen@juniper.net>";

  description "This module defines the format for a ZeroTouch ownership voucher, which is produced by Vendors, relayed by Bootstrap Servers, and consumed by devices. The purpose of the voucher is to enable a
```
device to ascertain the identity of its rightful owner, as
certified by its Vendor.

Copyright (c) 2014 IETF Trust and the persons identified as
authors of the code. All rights reserved.

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

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

revision "2015-10-19" {
  description
    "Initial version";
  reference
    "RFC XXXX: Zero Touch Provisioning for NETCONF Call Home";
}

// top-level container
container voucher {
  description
    "A voucher, containing the owner’s identifier, a list of
device’s unique identifiers, information on when the
voucher was created, when it might expire, and the
vendor’s signature over the above values.";
  leaf owner-id {
    type string;
    mandatory true;
    description
    "A Vendor-assigned value for the rightful owner of the
devices enumerated by this voucher. The owner-id value
must match the value in the owner-certificate below";
  }
  leaf-list unique-id {
    type string;
    min-elements 1;
    description
    "The unique identifier (e.g., serial-number) for a device.
The value must match the value in the device’s IDevID
certificate. A device uses this value to determine if
the voucher applies to it.";
  }
  leaf created-on {

type yang:date-and-time;
mandatory true;
description "The date this voucher was created";
}
leaf expires-on {
    type yang:date-and-time;
    description "The date this voucher expires, if at all. Use of this value requires that the device has access to a trusted real time clock";
}
leaf signature {
    type string;
    mandatory true;
    description "The signature over the concatenation of all the previous values";
}

Appendix B. Change Log

B.1. ID to 00

- Major structural update; the essence is the same. Most every section was rewritten to some degree.
- Added a Use Cases section
- Added diagrams for "Actors and Roles" and "NMS Precondition" sections, and greatly improved the "Device Boot Sequence" diagram
- Removed support for physical presence or any ability for Configlets to not be signed.
- Defined the Zero Touch Information DHCP option
- Added an ability for devices to also download images from Configuration Servers
- Added an ability for Configlets to be encrypted
Internet-Draft                 Zero Touch                   October 2015

- Now Configuration Servers only have to support HTTP/S - no other
  schemes possible

B.2.  00 to 01
- Added boot-image and validate-owner annotations to the "Actors and
  Roles" diagram.
- Fixed 2nd paragraph in section 7.1 to reflect current use of
  anyxml.
- Added encrypted and signed-encrypted examples
- Replaced YANG module with XSD schema
- Added IANA request for the Zero Touch Information DHCP Option
- Added IANA request for media types for boot-image and
  configuration

B.3.  01 to 02
- Replaced the need for a Configuration Signer with the ability for
  each NMS to be able to sign its own configurations, using
  Manufacturer signed Ownership Vouchers and Owner certificates.
- Renamed Configuration Server to Bootstrap Server, a more
  representative name given the information devices download from
  it.
- Replaced the concept of a Configlet by defining a southbound
  interface for the Bootstrap Server using YANG.
- Removed the IANA request for the boot-image and configuration
  media types

B.4.  02 to 03
- Minor update, mostly just to add an Editor’s Note to show how this
  draft might integrate with the draft-pritikin-anima-bootstrapping-
  keyinfra.

B.5.  03 to 04
- Major update formally introducing unsigned data and support for
  Internet-based redirect servers.
- Added many terms to Terminology section.
Added all new "Guiding Principles" section.

Added all new "Sources for Bootstrapping Data" section.

Rewrote the "Interactions" section and renamed it "Workflow Overview".

Authors’ Addresses

Kent Watsen
Juniper Networks
EMail: kwatsen@juniper.net

Joe Clarke
Cisco Systems
EMail: jclarke@cisco.com

Mikael Abrahamsson
T-Systems
EMail: "mikael.abrahamsson@t-systems.se"
Abstract

The registration policy for the Network Configuration Protocol (NETCONF) Capability URNs registry, set up by RFC 6241, has turned out to be unnecessarily strict. This document changes that registration policy to "IETF Review", allowing registrations from certain well reviewed Experimental RFCs, in addition to Standards Track RFCs.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 16, 2016.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

1. Introduction
The Network Configuration Protocol (NETCONF) Capability URNs registry [RFC6241], was set up with a registration policy of "Standards Action" [RFC5226], allowing registrations only from Standards Track RFCs. This provided thorough review of the specifications that are requesting NETCONF Capability URNs. It has turned out to be desirable to allocate capability URNs for certain Experimental RFCs also, provided those specifications are also carefully reviewed. The existing registration policy is, therefore, unnecessarily strict, requiring exception handling by the IESG. This document changes that registration policy to "IETF Review", which also allows registrations from certain well reviewed Experimental RFCs.

2. IANA Considerations

IANA is asked to change the registration policy for the Network Configuration Protocol (NETCONF) Capability URNs registry to "IETF Review", and to add this document to the registry’s reference field.

Registrations made from RFCs that are not on the Standards Track need to be carefully reviewed through IETF Last Call and consultation with relevant working groups, such as NETCONF. The Operations and Management Area Directors should confirm the appropriate level of review during IESG Evaluation.

3. Security Considerations

This document is purely procedural, and there are no related security considerations.

4. NOTE

[RFC Editor: Please remove this section and make sure the title of the references section is "Normative References". The version of xml2rfc that I’m using ignores my section title and just calls it "References".]

5. References


Author’s Address
Barry Leiba
Huawei Technologies

Phone: +1 646 827 0648
Email: barryleiba@computer.org
URI: http://internetmessagingtechnology.org/
Abstract

This document defines Restconf subscription and push mechanisms to continuously stream information from YANG datastores over HTTP. These mechanisms allow client applications or operations support systems to request custom sets of updates from a YANG datastore. This document also specifies how to stream updates over HTTP without Restconf. In either case, updates are pushed by a datastore to a receiver per a subscription policy, without requiring continuous requests.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 15, 2016.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of
1. Introduction

Requirements for subscriptions to YANG datastores are defined in [pub-sub-reqs]. Mechanisms to support YANG subscriptions and datastore object push over a NETCONF are defined in [netconf-yang-push]. Restconf support of subscriptions, with HTTP transport of pushed updates is also needed by the market. This document provides such a specification.

Key benefits of pushing data via HTTP include:

- Ability to configure static subscriptions on a Publisher


o Ability for the Publisher to initiate communications with the Receiver

o Ability of a Subscriber to be different from the Receiver

There are also additional benefits which can be realized when pushing updates via HTTP/2 [RFC7540]:

o Subscription multiplexing over independent HTTP/2 streams

o Stream prioritization

o Stream dependencies

o Flow control on independent streams

o Header compression

These additional benefits will address issues resulting from head-of-line blocking and relative subscription priority.

To maximize transport independence of YANG subscription methods, this document reuses many capabilities of [netconf-yang-push] including:

o Operations for creating, modifying and deleting subscriptions

o Syntax and parameters for negotiating the subscription

o YANG data model to manage subscriptions

o Mechanisms to communicate subscription filters and data streams

2. 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 RFC 2119 [RFC2119].

Datastore: a conceptual store of instantiated management information, with individual data items represented by data nodes which are arranged in hierarchical manner.

Dynamic YANG Subscription: Subscription negotiated with Publisher via create, modify, and delete control plane signaling messages.

Publisher: an entity responsible for distributing subscribed YANG object data per the terms of a Subscription. In general, a Publisher
is the owner of the YANG datastore that is subjected to the Subscription.

Receiver: the target where a Publisher pushes updates. In many deployments, the Receiver and Subscriber will be the same entity.

Static YANG Subscription: A Subscription installed via a configuration interface.

Subscriber: An entity able to request and negotiate a contract for push updates from a Publisher.

Subscription: A contract between a Subscriber and a Publisher, stipulating which information the Receiver wishes to have pushed from the Publisher without the need for further solicitation.

Subscription Update: Set of data nodes and object values pushed together as a unit and intended to meet the obligations of a single subscription at a snapshot in time.

3. Solution

This document specifies mechanisms that allow subscribed information updates to be pushed from a YANG datastore. Subscriptions may either be initiated via requests by Subscribers, or statically configured on a Publisher. As in [netconf-yang-push], Publisher must respond to a subscription request explicitly positively or negatively. Negative responses will include information about why the Subscription was not accepted, in order to facilitate converging on an acceptable set of Subscription parameters.

Once a Subscription has been established, updates are pushed to the Receiver until the Subscription terminates. Based on parameters within the Subscription, these updates can be streamed immediately as any subscribed objects change, or sent periodically.

3.1. Subscription Model

Subscriptions use the base data model from [netconf-yang-push]. This model is extended with several optional parameters for Subscription Priority and Subscription Dependency. These parameters allow a Subscriber or other configuration interface to assert how it prefers the Publisher allocate resources when handling multiple Subscriptions. These parameters are intended to be used in conjunction with the transport layer. Specifically, when a new Subscription is being established with an underlying transport is HTTP/2, these parameters may be directly mapped into HTTP/2 to
prioritize transport and to assist with flow control of individual streams.

3.2. Subscription states at Publisher

Below is the state machine for the Publisher. It is important to note that a Subscription doesn’t exist at the Publisher until it is accepted and made active. The assertion of a <create-subscription> by a Subscriber is insufficient for that asserted subscription to be externally visible via this state machine.

Subscription states at Publisher

Of interest in this state machine are the following:

- Successful <create-subscription> or <modify-subscription> actions must put the subscription into an active state.
- Failed <modify-subscription> actions will leave the subscription in its previous state, with no visible change to any streaming updates.
- A <delete-subscription> action will delete the entire subscription.
3.3. Mechanisms for Subscription Establishment and Maintenance

On a Publisher, it must be possible to instantiate a Subscription via dynamic Subscriber signaling, as well as via Static configuration.

Dynamic YANG Subscriptions are signaled Subscriptions aimed at the running datastore and are unable to impact the startup configuration. They should always terminate when there is loss of transport session connectivity between the Publisher and Receiver.

Static Subscriptions are applied via an operations interface to the startup and running configurations. Loss or non-availability of transport session connectivity will place the Subscription into the suspended state. Logic beyond the scope of this specification will dictate when any particular Subscription should be reactivated.

There are three models for Subscription establishment and maintenance:

1. Dynamic YANG Subscription: Subscriber and Receiver are the same
2. Static YANG Subscription
3. Dynamic YANG Subscription: Subscriber and Receiver are different

The first two are described in this section. The third is described in Appendix A. This third option can be moved into the body of this specification should the IETF community desire. In theory, all three models may be intermixed in a single deployment. Figure 2 shows such a scenario.
3.3.1. Dynamic YANG Subscription: Subscriber and Receiver are the same

With all Dynamic YANG Subscriptions, as with [netconf-yang-push] it must be possible to configure and manage Subscriptions via signaling. This signaling is transported over [restconf]. Once established, streaming Subscription Updates are then delivered via Restconf SSE.

3.3.2. Static YANG Subscription

With a Static YANG Subscription, all information needed to establish a secure object push relationship with that Receiver must be configured via a configuration interface on the Publisher. This information includes all the <create-subscription> information identified in section 3.3.1. This information also includes the Receiver address, encoding selection, and any security credentials required to establish TLS between the Publisher and Receiver. Mechanisms for locally configuring these parameters are outside the scope of this document.

With this information, the Publisher will establish a secure transport connection with the Receiver and then begin pushing the streaming updates to the Receiver. Since Restconf might not exist on the Receiver, it is not desirable to require that updates be pushed via Restconf. In place of Restconf, a TLS secured HTTP Client connection must be established with an HTTP Server located on the Receiver. Subscription Updates will then be sent via HTTP Post messages to the Receiver.

Post messages will be addressed to HTTP augmentation code on the Receiver capable accepting and responding to Subscription Updates. At least the initial Post message must include the URI for the subscribed resource. This URI can be retained for future use by the Receiver.

After successful receipt of an initial Subscription Update for a particular Subscription, this augmentation should reply back with an HTTP status code of 201 (Created). Further successful receipts should result in the return of code of 202 (Accepted). At any point, receipt of any status codes from 300-510 with the exception of 408 (Request Timeout) should result in the movement of the Subscription to the suspended state. A sequential series of multiple 408 exceptions should also drive the Subscription to a suspended state.

Security on an HTTP client/Publisher can be strengthened by only accepting Response code feedback for recently initiated HTTP POSTs.

Figure 3 depicts this message flow.
If HTTP/2 transport is available to a Receiver, the Publisher should also:

- point individual Subscription Updates to a unique HTTP/2 stream for that Subscription,
- take any subscription-priority and provision it into the HTTP/2 stream priority, and
- take any subscription-dependency and provision it into the HTTP/2 stream dependency.

### 3.4. Negotiation of Subscription Policies

When using signaling to create a Dynamic YANG Subscription, negotiable parameters will include the same negotiable parameters defined within [netconf-yang-push].

Additionally, negotiation may also include Subscription Priority. A Publisher may accept a Subscriber asserted Priority, as well as rejecting a subscription with a hint at what priority might be accepted.

### 3.5. Support for Periodic and On-change

Implementations must support periodic and/or on-change subscriptions as defined in [netconf-yang-push].
3.6. Filters and Streams

Implementations must support filters and streams as defined in [netconf-yang-push].

3.7. Authorization

Same authorization model for data as [netconf-yang-push] will be used. This includes functions of the Netconf Access Control Model [RFC6536] applied to objects to be pushed via Restconf.

A Subscription (including a Static YANG Subscription) may only be established if the Subscriber or some entity statically configuring via the Publisher’s operational interface has read access to the target data node.

3.8. Subscription Multiplexing

When pushed directly over HTTP/2, it is expected that each Subscription Update will be allocated a separate Stream. The will enable multiplexing, and address issues of Head-of-line blocking with different priority Subscriptions.

When pushed via Restconf over HTTP/2, different Subscriptions will not be mapped to independent HTTP/2 streams. When Restconf specifies this mapping, it should be integrated into this specification.

Even without HTTP/2 multiplexing, it is possible that updates might be delivered in a different sequence than generated. Reasons for this might include (but are not limited to):

- different durations needed to create various Subscription Updates,
- marshalling and bundling of multiple Subscription Updates for transport, and
- parallel HTTP1.1 sessions

Therefore each Subscription Update will include a microsecond level timestamp to ensure that a receiver understands the time when a that update was generated. Use of this timestamp can give an indication of the state of objects at a Publisher when state-entangled information is received across different subscriptions. The use of the latest Subscription Update timestamp for a particular object update can introduce errors. So when state-entangled updates have inconsistent object values and temporally close timestamps, a Receiver might consider performing a ‘get’ to validate the current state of objects.
3.9. Push Data Stream and Transport Mapping

Transported updates will contain data for one or more Subscription Updates. Each transported Subscription Update notification contains several parameters:

- A global subscription ID correlator, referencing the name of the Subscription on whose behalf the notification is sent.
- Data nodes containing a representation of the datastore subtree containing the updates. The set of data nodes must be filtered per access control rules to contain only data that the subscriber is authorized to see.
- An event time which contains the time stamp at publisher when the event is generated.

3.9.1. Pushing Subscription Updates via Restconf

Subscribers can dynamically learn whether a RESTCONF server supports yang-push. This is done by issuing an HTTP request OPTIONS, HEAD, or GET on the stream push-update. E.g.:

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

If the server supports it, it may respond

HTTP/1.1 200 OK
Content-Type: application/yang.api+xml
  <name>yang-push</name>
  <description>Yang push stream</description>
  <access>
    <encoding>xml</encoding>
    <location>https://example.com/streams/yang-push-xml</location>
  </access>
  <access>
    <encoding>json</encoding>
    <location>https://example.com/streams/yang-push-json</location>
  </access>
</stream>

If the server does not support yang push, it may respond
HTTP/1.1 404 Not Found
Date: Mon, 25 Apr 2012 11:10:30 GMT
Server: example-server

Subscribers can determine the URL to receive updates by sending an HTTP GET request for the "location" leaf with the stream list entry. The stream to use for yang push is the push-update stream. The location returned by the publisher can be used for the actual notification subscription. Note that different encodings are supporting using different locations. For example, he subscriber might send the following request:

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

The publisher might send the following response:

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

To subscribe and start receiving updates, the subscriber can then send an HTTP GET request for the URL returned by the publisher in the request above. The accept header must be "text/event-stream". The publisher handles the connection as an event stream, using the Server Sent Events[W3C-20121211] transport strategy.

The publisher MUST support as query parameters for a GET method on this resource all the parameters of a subscription. The only exception is the encoding, which is embedded in the URI. An example of this is:

// subtree filter = /foo
// periodic updates, every 5 seconds
GET /mystreams/yang-push?subscription-id=my-sub&period=5&
    xpath-filter=%2Fex:foo[starts-with("bar","some")]

Should the publisher not support the requested subscription, it may reply:
HTTP/1.1 501 Not Implemented
Date: Mon, 23 Apr 2012 17:11:00 GMT
Server: example-server
Content-Type: application/yang.errors+xml
<errors xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <error>
    <error-type>application</error-type>
    <error-tag>operation-not-supported</error-tag>
    <error-severity>error</error-severity>
    <error-message>Xpath filters not supported</error-message>
    <error-info>
      <supported-subscription xmlns="urn:ietf:params:xml:ns:
        netconf:datastore-push:1.0">
        <subtree-filter/>
      </supported-subscription>
    </error-info>
  </error>
</errors>

with an equivalent JSON encoding representation of:

HTTP/1.1 501 Not Implemented
Date: Mon, 23 Apr 2012 17:11:00 GMT
Server: example-server
Content-Type: application/yang.errors+json
{
  "ietf-restconf:errors": {
    "error": {
      "error-type": "protocol",
      "error-tag": "operation-not-supported",
      "error-message": "Xpath filters not supported."
      "error-info": {
        "datastore-push:supported-subscription": {
          "subtree-filter": [null]
        }
      }
    }
  }
}

The following is an example of a push Subscription Update data for the subscription above. It contains a subtree with root foo that contains a leaf called bar:
XML encoding representation:
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <subscription-id xmlns="urn:ietf:params:xml:ns:restconf:datastore-push:1.0">
    my-sub
  </subscription-id>
  <eventTime>2015-03-09T19:14:56Z</eventTime>
  <datastore-contents xmlns="urn:ietf:params:xml:ns:restconf:datastore-push:1.0">
    <foo xmlns="http://example.com/yang-push/1.0">
      <bar>some_string</bar>
    </foo>
  </datastore-contents>
</notification>

Or with the equivalent YANG over JSON encoding representation as defined in[yang-json] :

```
{
  "ietf-restconf:notification": {
    "datastore-push:subscription-id": "my-sub",
    "eventTime": "2015-03-09T19:14:56Z",
    "datastore-push:datastore-contents": {
      "example-mod:foo": { "bar": "some_string" }
    }
  }
}
```

To modify a subscription, the subscriber issues another GET request on the provided URI using the same subscription-id as in the original request. For example, to modify the update period to 10 seconds, the subscriber may send:

GET /mystreams/yang-push?subscription-id=my-sub&period=10&subtree-filter=%2Ffoo'

To delete a subscription, the subscriber issues a DELETE request on the provided URI using the same subscription-id as in the original request

DELETE /mystreams/yang-push?subscription-id=my-sub

3.9.2. Pushing Subscription Updates directly via HTTP

For any version of HTTP, the basic encoding will look as below is the above JSON representation wrapped in an HTTP header. Mechanism will be
POST (IP+Port) HTTP/1.1
From: (Identifier for Network Element)
User-Agent: (CiscoYANGPubSub/1.0)
Content-Type: multipart/form-data
Content-Length: (determined runtime)
{
    "ietf-yangpush:notification": {
        "datastore-push:subscription-id": "my-sub",
        "eventTime": "2015-03-09T19:14:56Z",
        "datastore-push:datastore-contents": {
            "foo": { "bar": "some_string" }
        }
    }
}

3.10.  YANG Tree

Below is the object tree for the model. All items are imported from [netconf-yang-push] except for the addition of "subscription-priority" and "subscription-dependency".

module: ietf-restconf-yang-push
  +ro system-streams
    |  +ro system-stream*
    +rw filters
      +rw filter* [filter-id]
      |  +rw filter-id
      |  +rw subtree-filter?
      |  +rw xpath-filter? yang:xpath1.0
    +rw subscription-config
      +rw datastore-push-subscription* [subscription-id]
        +rw datastore-push-subscription* [subscription-id]
          +rw subscription-id
          +rw target-datastore? datastore
          +rw stream? system-stream
          +rw encoding? encoding
          +rw start-time? yang:date-and-time
          +rw stop-time? yang:date-and-time
          +rw (update-trigger)?
            +-(periodic)
              |  +rw period? yang:timeticks
            +-(on-change)
              |  +rw no-synch-on-start? empty
              |  +rw dampening-period yang:timeticks
              |  +rw excluded-change* change-type
            +rw (filterspec)?
              |  +-(inline)
              |    |  +rw subtree-filter? subtree-filter

4. YANG Module


prefix "rc-push";

import ietf-datastore-push {
    prefix ds-push;
}

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

contact
"WG Web:  <http://tools.ietf.org/wg/netconf/>
WG List:  <mailto:netconf@ietf.org>
WG Chair: Mahesh Jethanandani
<mailto:mjethanandani@gmail.com >
WG Chair: Mehmet Ersue
<mailto:mehmet.ersue@nokia.com>
Editor:   Eric Voit
<mailto:evoit@cisco.com>
Editor:   Alexander Clemm
<mailto:alex@cisco.com>
Editor:   Ambika Prasad Tripathy
<mailto:ambtripa@cisco.com>
Editor:   Einar Nilsen-Nygaard
<mailto:einar.nn@cisco.com>
Editor:   Alberto Gonzalez Prieto
<mailto:albertgo@cisco.com>";

description
"This module contains conceptual YANG specifications for
Restconf datastore push."

revision 2015-10-01 {
    description
        "Initial revision."
    reference "restconf YANG Datastore push"
}

grouping subscription-qos {
    description
        "This grouping describes Quality of Service information
        concerning a subscription. This information is passed to lower
        layers for transport prioritization and treatment"
    leaf subscription-priority {
        
type uint8;
description
"Relative priority for a subscription. Allows an underlying transport layer perform informed load balance allocations between various subscriptions";
}
leaf subscription-dependency {
type string;
description
"Provides the Subscription ID of a parent subscription without which this subscription should not exist. In other words, there is no reason to stream these objects if another subscription is missing.";
}
}
augment "/ds-push:subscription-config/" +
"ds-push:datastore-push-subscription" {
description
"Augments configured subscriptions with QoS parameters.";
uses subscription-qos;
}

augment "/ds-push:subscriptions/" +
"ds-push:datastore-push-subscription" {
description
"Augments the list of currently active subscriptions with QoS parameters.";
uses subscription-qos;
}

augment "/ds-push:create-subscription/" +
"ds-push:input" {
description
"Augments the create subscription rpc with QoS parameters.";
uses subscription-qos;
}

augment "/ds-push:modify-subscription/" +
"ds-push:input" {
description
"Augments the modify subscription rpc with QoS parameters.";
uses subscription-qos;
}
5. Security Considerations

Subscriptions could be used to intentionally or accidentally overload resources of a Publisher. For this reason, it is important that the Publisher has the ability to prioritize the establishment and push of updates where there might be resource exhaust potential. In addition, a server needs to be able to suspend existing subscriptions when needed. When this occurs, the subscription status must be updated accordingly and the clients are notified.

A Subscription could be used to retrieve data in subtrees that a client has not authorized access to. Therefore it is important that data pushed via a Subscription is authorized equivalently with regular data retrieval operations. Data being pushed to a client needs therefore to be filtered accordingly, just like if the data were being retrieved on-demand. The Netconf Authorization Control Model [RFC6536] applies.

One or more Publishers could be used to overwhelm a Receiver which doesn’t even support subscriptions. Therefore Updates MUST only be transmittable over Encrypted transports. Clients which do not want pushed data need only terminate or refuse any transport sessions from the Publisher.

One or more Publishers could overwhelm a Receiver which is unable to control or handle the volume of Updates received. In deployments where this might be a concern, transports supporting per-subscription Flow Control and Prioritization (such as HTTP/2) should be selected.

Another benefit is that a well-behaved Publisher implementation is that it is difficult to a Publisher to perform a DoS attack on a Receiver. DoS attack protection comes from:

- the requirement for trust of a TLS session before publication,
- the need for an HTTP transport augmentation on the Receiver, and
- that the Publication process is suspended when the Receiver doesn’t respond.

6. References

6.1. Normative References


6.2. Informative References


Appendix A. Dynamic YANG Subscription when the Subscriber and Receiver are different

The methods of Sections 3.3.1 and 3.3.2 can be combined to enable deployment models where the Subscriber and Receiver are different. Such separation can be useful with some combination of:

- An operator wants any Subscriptions immediately deleted should TLS connectivity be lost. (I.e., Subscriptions don’t default into a ‘Suspended’ state on the Publisher.)
- An operator wants the Publisher to include highly restrictive capacity management and security mechanisms outside of domain of existing operational or programmatic interfaces.
- Restconf is not desired on the Receiver.
- The Publisher doesn’t want to maintain Restconf subscriptions with many Receivers.

To do this, first the necessary information must be signaled as part of the <create-subscription>. This includes all the information described in section 3.3.2, with the exception of the security credentials. (It is assumed that any security credentials required for establishing any transport connections are pre-provisioned on all devices.)

Using this set of Subscriber provided information, the same process described within section 3.3.2 will be followed. There is one exception. When an HTTP status code is 201 is received by the Publisher, it will inform the Subscriber of Subscription establishment success via its Restconf connection.

After successful establishment, if the Subscriber wishes to maintain the state of Receiver subscriptions, it can simply place a separate on-change Subscription into the "Subscriptions" subtree of the YANG datastore on the Publisher.

Putting it all together, the message flow is:
Appendix B. End-to-End Deployment Guidance

Several technologies are expected to be seen within a deployment to achieve security and ease-of-use requirements. These are not necessary for an implementation of this specification, but will be useful to consider when considering the operational context.

B.1. Call Home

Pub/Sub implementations should have the ability to transparently incorporate lower layer technologies such as Call Home so that secure TLS connections are always originated from the Publisher. There is a Restconf Call home function in [call-home]. For security reasons, this should be implemented as desired.

B.2. TLS Heartbeat

Unlike NETCONF, HTTP sessions might not quickly allow a Subscriber to recognize when the communication path has been lost from the Publisher. To recognize this, it is possible for a Receiver (usually the subscriber) to establish a TLS heartbeat [RFC6520]. In the case where a TLS heartbeat is included, it should be sent just from Receiver to Publisher. Loss of the heartbeat should result in the Subscription being terminated with the Subscriber (even when the
Subscriber and Receiver are different). The Subscriber can then attempt to re-establish the subscription if desired. If the Subscription remains active on the Publisher, future receipt of objects associated with that (or any other unknown) subscription ID should result in a <delete-subscription> being returned to the Publisher from the Receiver.

B.3. Putting it together

If Subscriber and receiver are same entity then subscriber can direct send create_subscription message to publisher. Once the subscription moved to accepted state, the receiver can use Server Sent Events [W3C-20121211] transport strategy to subscriber event notifications for the data as defined in[restconf].

Authors’ Addresses

Eric Voit
Cisco Systems
Email: evoit@cisco.com

Alexander Clemm
Cisco Systems
Email: alex@cisco.com

Ambika Prasad Tripathy
Cisco Systems
Email: ambtripa@cisco.com

Einar Nilsen-Nygaard
Cisco Systems
Email: einarnn@cisco.com

Alberto Gonzalez Prieto
Cisco Systems
Email: albertgo@cisco.com