

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: April 30, 2017

A. Bierman  
YumaWorks  
M. Bjorklund  
Tail-f Systems  
K. Watsen  
Juniper Networks  
October 27, 2016

RESTCONF Protocol  
draft-ietf-netconf-restconf-18

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on April 30, 2017.

Copyright Notice

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

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

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

## Table of Contents

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

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

11.3.2. Media Type application/yang-data+json . . . . .	86
11.4. RESTCONF Capability URNs . . . . .	88
11.5. Registration of "restconf" URN sub-namespace . . . . .	89
12. Security Considerations . . . . .	89
13. Acknowledgements . . . . .	90
14. References . . . . .	91
14.1. Normative References . . . . .	91
14.2. Informative References . . . . .	94
Appendix A. Change Log . . . . .	94
A.1. v17 to v18 . . . . .	94
A.2. v16 to v17 . . . . .	94
A.3. v15 to v16 . . . . .	95
A.4. v14 to v15 . . . . .	95
A.5. v13 - v14 . . . . .	96
A.6. v12 - v13 . . . . .	98
A.7. v11 - v12 . . . . .	98
A.8. v10 - v11 . . . . .	98
A.9. v09 - v10 . . . . .	99
A.10. v08 - v09 . . . . .	101
A.11. v07 - v08 . . . . .	101
A.12. v06 - v07 . . . . .	102
A.13. v05 - v06 . . . . .	102
A.14. v04 - v05 . . . . .	102
A.15. v03 - v04 . . . . .	103
A.16. v02 - v03 . . . . .	103
A.17. v01 - v02 . . . . .	104
A.18. v00 - v01 . . . . .	105
A.19. bierman:restconf-04 to ietf:restconf-00 . . . . .	106
Appendix B. Open Issues . . . . .	106
Appendix C. Example YANG Module . . . . .	106
C.1. example-jukebox YANG Module . . . . .	107
Appendix D. RESTCONF Message Examples . . . . .	112
D.1. Resource Retrieval Examples . . . . .	112
D.1.1. Retrieve the Top-level API Resource . . . . .	112
D.1.2. Retrieve The Server Module Information . . . . .	113
D.1.3. Retrieve The Server Capability Information . . . . .	115
D.2. Edit Resource Examples . . . . .	116
D.2.1. Create New Data Resources . . . . .	116
D.2.2. Detect Resource Entity-Tag Change . . . . .	117
D.2.3. Edit a Datastore Resource . . . . .	118
D.2.4. Replace a Datastore Resource . . . . .	119
D.2.5. Edit a Data Resource . . . . .	120
D.3. Query Parameter Examples . . . . .	120
D.3.1. "content" Parameter . . . . .	120
D.3.2. "depth" Parameter . . . . .	124
D.3.3. "fields" Parameter . . . . .	127
D.3.4. "insert" Parameter . . . . .	128
D.3.5. "point" Parameter . . . . .	129

D.3.6. "filter" Parameter . . . . .	130
D.3.7. "start-time" Parameter . . . . .	131
D.3.8. "stop-time" Parameter . . . . .	131
D.3.9. "with-defaults" Parameter . . . . .	132
Authors' Addresses . . . . .	133

## 1. Introduction

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

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

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

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

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

Configuration data and state data are exposed as resources that can be retrieved with the GET method. Resources representing configuration data can be modified with the DELETE, PATCH, POST, and PUT methods. Data is encoded with either XML [W3C.REC-xml-20081126] or JSON [RFC7159].

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

## 1.1. Terminology

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

### 1.1.1. NETCONF

The following terms are defined in [RFC6241]:

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

### 1.1.2. HTTP

The following terms are defined in [RFC3986]:

- o fragment
- o path
- o query

The following terms are defined in [RFC7230]:

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

The following terms are defined in [RFC7231]:

- o method
- o request
- o resource

The following terms are defined in [RFC7232]:

- o entity-tag

#### 1.1.3. YANG

The following terms are defined in [RFC7950]:

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

#### 1.1.4. NETCONF Notifications

The following terms are defined in [RFC5277]:

- o notification replay

## 1.1.1.5. Terms

The following terms are used within this document:

- o API resource: the resource that models the RESTCONF root resource and the sub-resources to access YANG-defined content. It is defined with the YANG data template named "yang-api" in the "ietf-restconf" module.
- o client: a RESTCONF client
- o data resource: a resource that models a YANG data node. It is defined with YANG data definition statements.
- o datastore resource: the resource that models a programmatic interface using NETCONF datastore concepts. By default, RESTCONF methods access a unified view of the underlying datastore implementation on the server. It is defined as a sub-resource within the API resource.
- o edit operation: a RESTCONF operation on a data resource using either a POST, PUT, PATCH, or DELETE method. This is not the same as the NETCONF edit operation (i.e., one of the values for the "nc:operation" attribute: "create", "replace", "merge", "delete", or "remove").
- o event stream resource: This resource represents an SSE (Server-Sent Events) event stream. The content consists of text using the media type "text/event-stream", as defined by the SSE [W3C.REC-eventsourcing-20150203] specification. Event stream contents are described in Section 3.8.
- o media-type: HTTP uses Internet media types [RFC2046] in the Content-Type and Accept header fields in order to provide open and extensible data typing and type negotiation.
- o NETCONF client: a client which implements the NETCONF protocol. Called "client" in [RFC6241].
- o NETCONF server: a server which implements the NETCONF protocol. Called "server" in [RFC6241].
- o operation: the conceptual RESTCONF operation for a message, derived from the HTTP method, request URI, header fields, and message-body.



- o operation resource: a resource that models a data-model-specific operation, that is defined with a YANG "rpc" or "action" statement. It is invoked with the POST method.
- o patch: a PATCH method on the target datastore or data resource. The media type of the message-body content will identify the patch type in use.
- o plain patch: a specific media type for use with the PATCH method, defined in Section 4.6.1, that can be used for simple merge operations. It is specified by a request Content-Type of "application/yang-data+xml" or "application/yang-data+json".
- o query parameter: a parameter (and its value if any), encoded within the query component of the request URI.
- o resource type: one of the RESTCONF resource classes defined in this document. One of "api", "datastore", "data", "operation", "schema", or "event stream".
- o RESTCONF capability: An optional RESTCONF protocol feature supported by the server, which is identified by an IANA registered NETCONF Capability URI, and advertised with an entry in the "capability" leaf-list defined in Section 9.3.
- o RESTCONF client: a client which implements the RESTCONF protocol.
- o RESTCONF server: a server which implements the RESTCONF protocol.
- o retrieval request: a request using the GET or HEAD methods.
- o schema resource: a resource that used by the client to retrieve a YANG schema with the GET method. It has a representation with the media type "application/yang".
- o server: a RESTCONF server
- o stream list: the set of data resource instances that describe the event stream resources available from the server. This information is defined in the "ietf-restconf-monitoring" module as the "stream" list. It can be retrieved using the target resource "{+restconf}/data/ietf-restconf-monitoring:restconf-state/streams/stream". The stream list contains information about each stream, such as the URL to retrieve the event stream data.
- o stream resource: An event stream resource.

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

#### 1.1.1.6. URI Template and Examples

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

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

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

#### 1.1.1.7. Tree Diagrams

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

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

### 1.2. Subset of NETCONF Functionality

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

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

The HTTP POST, PUT, PATCH, and DELETE methods are used to edit data resources represented by YANG data models. These basic edit operations allow the running configuration to be altered by a RESTCONF client.

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

### 1.3. Data Model Driven API

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

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

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

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

The RESTCONF protocol operates on a conceptual datastore defined with the YANG data modeling language. The server lists each YANG module it supports using the "ietf-yang-library" YANG module, defined in [RFC7895]. The server MUST implement the "ietf-yang-library" module, which MUST identify all the YANG modules used by the server, in the "modules-state/module" list. The conceptual datastore contents, data-model-specific RPC operations and event notifications are identified by this set of YANG modules.

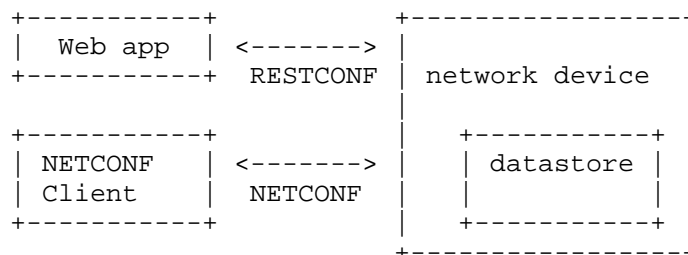
The classification of data as configuration or non-configuration is derived from the YANG "config" statement. Data ordering behavior is derived from the YANG "ordered-by" statement. Non-configuration data is also called "state data".

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

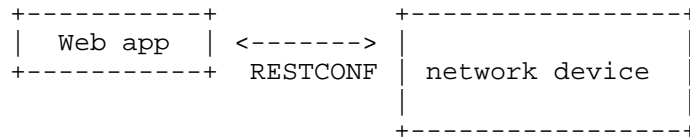
#### 1.4. Coexistence with NETCONF

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

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



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



There are interactions between the NETCONF protocol and RESTCONF protocol related to edit operations. It is possible that locks are in use on a RESTCONF server, even though RESTCONF cannot manipulate locks. In such a case, the RESTCONF protocol will not be granted write access to data resources within a datastore.

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

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

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

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

## 1.5. RESTCONF Extensibility

There are two extensibility mechanisms built into RESTCONF:

- o protocol version

- o optional capabilities

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

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

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

The client might send:

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

The server might respond:

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

<XRD xmlns='http://docs.oasis-open.org/ns/xri/xrd-1.0'>
  <Link rel='restconf' href='/restconf'/>
  <Link rel='restconf2' href='/restconf2'/>
</XRD>
```

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

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

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

## 2. Transport Protocol

### 2.1. Integrity and Confidentiality

HTTP [RFC7230] is an application layer protocol that may be layered on any reliable transport-layer protocol. RESTCONF is defined on top of HTTP, but due to the sensitive nature of the information conveyed, RESTCONF requires that the transport-layer protocol provides both data integrity and confidentiality. A RESTCONF server **MUST** support the TLS protocol [RFC5246], and **SHOULD** adhere to [RFC7525]. The RESTCONF protocol **MUST NOT** be used over HTTP without using the TLS protocol.

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

### 2.2. HTTPS with X.509v3 Certificates

Given the nearly ubiquitous support for HTTP over TLS [RFC7230], RESTCONF implementations **MUST** support the "https" URI scheme, which has the IANA assigned default port 443.

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

### 2.3. Certificate Validation

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

### 2.4. Authenticated Server Identity

The RESTCONF client **MUST** check the identity of the server according to Section 3.1 of [RFC2818].

## 2.5. Authenticated Client Identity

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

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

A server MAY also support the combination of both client certificates and an HTTP client authentication scheme, with the determination of how to process this combination left as an implementation decision.

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

## 3. Resources

The RESTCONF protocol operates on a hierarchy of resources, starting with the top-level API resource itself (Section 3.1). Each resource represents a manageable component within the device.

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

A resource has a representation associated with a media type identifier, as represented by the "Content-Type" header field in the HTTP response message. A resource has one or more representations, each associated with a different media type. When a representation of a resource is sent in an HTTP message, the associated media type is given in the "Content-Type" header. A resource can contain zero or more nested resources. A resource can be created and deleted independently of its parent resource, as long as the parent resource exists.



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

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

### 3.1. Root Resource Discovery

In line with the best practices defined by [RFC7320], RESTCONF enables deployments to specify where the RESTCONF API is located. When first connecting to a RESTCONF server, a RESTCONF client MUST determine the root of the RESTCONF API. There MUST be exactly one "restconf" link relation returned by the device.

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

Example returning `/restconf`:

The client might send:

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

The server might respond:

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

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

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

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

Example returning `/top/restconf`:

The client might send:

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

The server might respond:

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

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

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

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

The client might send:

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

The server might respond:

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

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

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

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

### 3.2. RESTCONF Media Types

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

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

### 3.3. API Resource

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

YANG Tree Diagram for an API Resource:

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

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

The API resource can be retrieved with the GET method.

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

This resource has the following child resources:

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

## RESTCONF API Resource

## 3.3.1. {+restconf}/data

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

Example:

This example request by the client would retrieve only the non-configuration data nodes that exist within the "library" resource, using the "content" query parameter (see Section 4.8.1).

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

The server might respond:

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

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

## 3.3.2. {+restconf}/operations

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

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

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

Operation resources are defined in Section 3.6.

### 3.3.3. `{+restconf}/yang-library-version`

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

Example:

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

The server might respond:

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

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

### 3.4. Datastore Resource

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

This resource type is an abstraction of the system's underlying datastore implementation. The client uses it to edit and retrieve data resources, as the conceptual root of all configuration and state data that is present on the device.

Configuration edit transaction management and configuration persistence are handled by the server and not controlled by the

client. A datastore resource can be written directly with the POST and PATCH methods. Each RESTCONF edit of a datastore resource is saved to non-volatile storage by the server, if the server supports non-volatile storage of configuration data, as described in Section 1.4.

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

#### 3.4.1. Edit Collision Prevention

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

##### 3.4.1.1. Timestamp

The last change time is maintained and the "Last-Modified" ([RFC7232], Section 2.2) header field is returned in the response for a retrieval request. The "If-Unmodified-Since" header field ([RFC7232], Section 3.4) can be used in edit operation requests to cause the server to reject the request if the resource has been modified since the specified timestamp.

The server SHOULD maintain a last-modified timestamp for the datastore resource, defined in Section 3.4. This timestamp is only affected by configuration child data resources, and MUST NOT be updated for changes to non-configuration child data resources. Last-modified timestamps for data resources are discussed in Section 3.5.

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

##### 3.4.1.2. Entity-Tag

The server MUST maintain a unique opaque entity-tag for the datastore resource and MUST return it in the "ETag" ([RFC7232], Section 2.3) header in the response for a retrieval request. The client MAY use an "If-Match" header in edit operation requests to cause the server

to reject the request if the resource entity-tag does not match the specified value.

The server MUST maintain an entity-tag for the top-level `{+restconf}/data` resource. This entity-tag is only affected by configuration data resources, and MUST NOT be updated for changes to non-configuration data. Entity-tags for data resources are discussed in Section 3.5. Note that each representation (e.g., XML vs. JSON) requires a different entity-tag.

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

#### 3.4.1.3. Update Procedure

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

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

#### 3.5. Data Resource

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

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

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

### 3.5.1. Timestamp

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

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

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

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

### 3.5.2. Entity-Tag

For configuration data resources, the server SHOULD maintain a resource entity-tag for each resource, and return the "ETag" header field when it is retrieved as the target resource with the GET or HEAD methods. If maintained, the resource entity-tag MUST be updated whenever the resource or any configuration resource within the resource is altered. If not maintained, then the resource entity-tag for the datastore MUST be used instead.

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

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

### 3.5.3. Encoding Data Resource Identifiers in the Request URI

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



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

A RESTCONF data resource identifier is encoded from left to right, starting with the top-level data node, according to the "api-path" rule in Section 3.5.3.1. The node name of each ancestor of the target resource node is encoded in order, ending with the node name for the target resource. If a node in the path is defined in another module than its parent node, or its parent is the datastore, then the module name followed by a colon character (":") MUST be prepended to the node name in the resource identifier. See Section 3.5.3.1 for details.

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

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

If a data node in the path expression is a YANG list node, then the key values for the list (if any) MUST be encoded according to the following rules:

- o The key leaf values for a data resource representing a YANG list MUST be encoded using one path segment [RFC3986].
- o If there is only one key leaf value, the path segment is constructed by having the list name, followed by an "=" character, followed by the single key leaf value.

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

Examples:

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

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

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

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

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

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

Example URL:

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

#### 3.5.3.1. ABNF For Data Resource Identifiers

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

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

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

#### 3.5.4. Default Handling

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

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

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

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

#### 3.6. Operation Resource

An operation resource represents an RPC operation defined with the YANG "rpc" statement or a data-model-specific action defined with a

YANG "action" statement. It is invoked using a POST method on the operation resource.

An RPC operation is invoked as:

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

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

For example, if "module-A" defined a "reset" rpc operation, then invoking the operation would be requested as follows:

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

An action is invoked as:

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

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

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

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

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

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

#### 3.6.1. Encoding Operation Resource Input Parameters

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

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

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

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

Examples:

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

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

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

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

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

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

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

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

  revision "2016-03-10";

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

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

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

RPC Input Example:



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

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

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

The server might respond:

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

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

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

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

Action Input Example:

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

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

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

The server might respond:

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

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

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

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

### 3.6.2. Encoding Operation Resource Output Parameters

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

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

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

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

Examples:

RPC Output Example:

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

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

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

The server might respond:

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

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

The same response is shown here using XML encoding:

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

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

Action Output Example:

The "example-actions" YANG module defined in Section 3.6.1 is used for this example.

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

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

The server might respond:

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

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

### 3.6.3. Encoding Operation Resource Errors

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

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

Using the "reboot" RPC operation from the example in Section 3.6.1, the client might send the following POST request message:

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

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

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

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

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

The same response is shown here in JSON encoding:

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

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

### 3.7. Schema Resource

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

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

The client might send the following GET request message:

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

The server might respond:

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

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

Next the client needs to retrieve the actual YANG schema.

The client might send the following GET request message:

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

The server might respond:

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

module example-jukebox {

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

}
```

### 3.8. Event Stream Resource

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

An event stream functions according to the NETCONF Notifications specification [RFC5277]. The available streams can be retrieved from the stream list, which specifies the syntax and semantics of the stream resources.

### 3.9. Errors YANG Data Template

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

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

### 4. RESTCONF Methods

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

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

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

#### CRUD Methods in RESTCONF

The "remove" edit operation attribute for the NETCONF <edit-config> RPC operation is not supported by the HTTP DELETE method. The resource must exist or the DELETE method will fail. The PATCH method is equivalent to a "merge" edit operation when using a plain patch (see Section 4.6.1); other media-types may provide more granular control.

Access control mechanisms are used to limit what CRUD operations can be used. In particular, RESTCONF is compatible with the NETCONF Access Control Model (NACM) [RFC6536], as there is a specific mapping between RESTCONF and NETCONF operations. The resource path needs to

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

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

Implementation of all methods (except PATCH [RFC5789]) are defined in [RFC7231]. This section defines the RESTCONF protocol usage for each HTTP method.

#### 4.1. OPTIONS

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

The "Accept-Patch" header field MUST be supported and returned in the response to the OPTIONS request, as defined in [RFC5789].

#### 4.2. HEAD

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

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

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

#### 4.3. GET

The RESTCONF server MUST support the GET method. The GET method is sent by the client to retrieve data and metadata for a resource. It is supported for all resource types, except operation resources. The request MUST contain a request URI that contains at least the root resource.

The server MUST NOT return any data resources for which the user does not have read privileges. If the user is not authorized to read the target resource, an error response containing a "401 Unauthorized"



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

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

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

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

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

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

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

Example:

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

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

The server might respond:

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

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

Refer to Appendix D.1 for more resource retrieval examples.

#### 4.4. POST

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

Type	Description
Datastore	Create a top-level configuration data resource
Data	Create a configuration data child resource
Operation	Invoke an RPC operation

Resource Types that Support POST

##### 4.4.1. Create Resource Mode

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

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

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

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

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

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

Example:

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

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

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

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

```
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2016 17:01:00 GMT
Server: example-server
Location: https://example.com/restconf/data/\
    example-jukebox:jukebox
Last-Modified: Mon, 23 Apr 2016 17:01:00 GMT
ETag: "b3a3e673be2"
```

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

#### 4.4.2. Invoke Operation Mode

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

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

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

Example:

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

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

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

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

The server might respond:

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

#### 4.5. PUT

The RESTCONF server MUST support the PUT method. The PUT method is sent by the client to create or replace the target data resource. A request message-body MUST be present, representing the new data

resource, or the server MUST return "400 Bad Request" status-line. The error-tag value "invalid-value" is used in this case.

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

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

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

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

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

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

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

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

Example:

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

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

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

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

If the resource is updated, the server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2016 17:04:00 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2016 17:04:00 GMT
ETag: "b27480aeda4c"
```

The same request is shown here using XML encoding:

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

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

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

#### 4.6. PATCH

The RESTCONF server MUST support the PATCH method for a plain patch, and MAY support additional media types. The PATCH media types supported by a server can be discovered by the client by sending an

OPTIONS request, and examining the Accept-Patch header field in the response. (see Section 4.1).

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

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

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

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

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

#### 4.6.1. Plain Patch

The plain patch mechanism merges the contents of the message-body with the target resource. The message-body for a plain patch MUST be present and MUST be represented by the media type "application/yang-data+xml" or "application/yang-data+json".

Plain patch can be used to create or update, but not delete, a child resource within the target resource. Please see [I-D.ietf-netconf-yang-patch] for an alternate media-type supporting the ability to delete child resources. The YANG Patch Media Type allows multiple sub-operations (e.g., merge, delete) within a single PATCH method.

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

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

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

Example:

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

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

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

If the field is updated, the server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2016 17:49:30 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2016 17:49:30 GMT
ETag: "b2788923da4c"
```

#### 4.7. DELETE

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

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

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

Example:



To delete the "album" resource with the key "Wasting Light", the client might send:

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

If the resource is deleted, the server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2016 17:49:40 GMT
Server: example-server
```

#### 4.8. Query Parameters

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

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

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

#### RESTCONF Query Parameters

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

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

##### 4.8.1. The "content" Query Parameter

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

The allowed values are:

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

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

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

#### 4.8.2. The "depth" Query Parameter

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

The requested data node has a depth level of '1'. If the "fields" parameter (Section 4.8.3) is used to select descendant data nodes, then these nodes and all their ancestor nodes have a depth value of 1. (This has the effect of including the nodes specified by the fields, even if the "depth" value is less than the actual depth level of the specified fields.) Any other child node has a depth value that is 1 greater than its parent.

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

This parameter is only allowed for GET methods on API, datastore, and data resources. A "400 Bad Request" status-line is returned if it used for other methods or resource types.

By default, the server will include all sub-resources within a retrieved resource, which have the same resource type as the requested resource. The exception is the datastore resource. If this resource type is retrieved then by default the datastore and all child data resources are returned.

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

#### 4.8.3. The "fields" Query Parameter

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

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

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

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

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

"api-identifier" is defined in Section 3.5.3.1.

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

Parentheses are used to specify sub-selectors of a node. Note that there is no path separator character '/' between a "path" field and left parenthesis character '('.

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

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

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

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

#### 4.8.4. The "filter" Query Parameter

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

This parameter is only allowed for GET methods on an event stream resource. A "400 Bad Request" status-line is returned if used for other methods or resource types.

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

- o The set of namespace declarations is the set of prefix and namespace pairs for all supported YANG modules, where the prefix is the YANG module name, and the namespace is as defined by the "namespace" statement in the YANG module.
- o The function library is the core function library defined in XPath 1.0, plus any functions defined by the data model.
- o The set of variable bindings is empty.
- o The context node is the root node.

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

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

#### 4.8.5. The "insert" Query Parameter

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

The allowed values are:

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

The default value is "last".

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

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

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

#### 4.8.6. The "point" Query Parameter

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

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

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

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

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

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

#### 4.8.7. The "start-time" Query Parameter

The "start-time" parameter is used to trigger the notification replay feature defined in [RFC5277] and indicate that the replay should start at the time specified. If the stream does not support replay, per the "replay-support" attribute returned by stream list entry for the stream resource, then the server MUST return a "400 Bad Request" status-line.

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

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

If this parameter is not present, then a replay subscription is not being requested. It is not valid to specify start times that are

later than the current time. If the value specified is earlier than the log can support, the replay will begin with the earliest available notification. A client can obtain a server's current time by examining the "Date" header field that the server returns in response messages, according to [RFC7231].

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

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

#### 4.8.8. The "stop-time" Query Parameter

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

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

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

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

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

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

#### 4.8.9. The "with-defaults" Query Parameter

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

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

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

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

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

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

If the "with-defaults" parameter is set to "report-all-tagged" then the server MUST adhere to the defaults reporting behavior defined in Section 3.4 of [RFC6243]. Metadata is reported by the server as specified in Section 5.3. The XML encoding for the "default" attribute sent by the server for default nodes is defined in section 6 of [RFC6243]. The JSON encoding for the "default" attribute MUST use the same values as defined in [RFC6243], but encoded according to the rules in [RFC7952]. The module name "ietf-netconf-with-defaults" MUST be used for the "default" attribute.

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

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



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

## 5. Messages

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

### 5.1. Request URI Structure

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

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

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

^	^	^	^
method	entry	resource	query
M	M	O	O

M=mandatory, O=optional

where:

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

- o method: the HTTP method identifying the RESTCONF operation requested by the client, to act upon the target resource specified in the request URI. RESTCONF operation details are described in Section 4.

- o entry: the root of the RESTCONF API configured on this HTTP server, discovered by getting the `"/.well-known/host-meta"` resource, as described in Section 3.1.
- o resource: the path expression identifying the resource that is being accessed by the RESTCONF operation. If this field is not present, then the target resource is the API itself, represented by the YANG data template named `"yang-api"`, found in Section 8.
- o query: the set of parameters associated with the RESTCONF message, as defined in section 3.4 of [RFC3986]. RESTCONF parameters have the familiar form of `"name=value"` pairs. Most query parameters are optional to implement by the server and optional to use by the client. Each optional query parameter is identified by a URI. The server MUST list the optional query parameter URIs it supports in the `"capabilities"` list defined in Section 9.3.

There is a specific set of parameters defined, although the server MAY choose to support query parameters not defined in this document. The contents of the any query parameter value MUST be encoded according to [RFC3986], Section 3.4. Any reserved characters MUST be percent-encoded, according to [RFC3986], section 2.1 and 2.5.

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

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

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

Refer to Appendix D for examples of RESTCONF Request URIs.

## 5.2. Message Encoding

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

Content is encoded in either JSON or XML format. A server MUST support one of either XML or JSON encoding. A server MAY support both XML and JSON encoding. A client will need to support both XML and JSON to interoperate with all RESTCONF servers.

XML encoding rules for data nodes are defined in [RFC7950]. The same encoding rules are used for all XML content. JSON encoding rules are defined in [RFC7951]. Additional JSON encoding rules for metadata are defined in [RFC7952]. This encoding is valid JSON, but also has special encoding rules to identify module namespaces and provide consistent type processing of YANG data.

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

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

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

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

### 5.3. RESTCONF Metadata

The RESTCONF protocol needs to support retrieval of the same metadata that is used in the NETCONF protocol. Information about default leaves, last-modified timestamps, etc. are commonly used to annotate representations of the datastore contents.

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

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

### 5.3.1. XML Metadata Encoding Example

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

The server might respond as follows.

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

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

### 5.3.2. JSON Metadata Encoding Example

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

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

The server might respond as follows.

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

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

#### 5.4. Return Status

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

#### 5.5. Message Caching

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

The server MUST include a "Cache-Control" header field in every response that specifies whether the response should be cached.

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

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

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

## 6. Notifications

The RESTCONF protocol supports YANG-defined event notifications. The solution preserves aspects of NETCONF Event Notifications [RFC5277] while utilizing the Server-Sent Events [W3C.REC-eventsource-20150203] transport strategy.

### 6.1. Server Support

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

### 6.2. Event Streams

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

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

For example:

The client might send the following request:

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

The server might send the following response:

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

```
<streams
  xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring">
  <stream>
    <name>NETCONF</name>
    <description>default NETCONF event stream</description>
    <replay-support>true</replay-support>
    <replay-log-creation-time>\
      2007-07-08T00:00:00Z\
    </replay-log-creation-time>
    <access>
      <encoding>xml</encoding>
      <location>https://example.com/streams/NETCONF\
    </location>
    </access>
    <access>
      <encoding>json</encoding>
      <location>https://example.com/streams/NETCONF-JSON\
    </location>
    </access>
  </stream>
  <stream>
    <name>SNMP</name>
    <description>SNMP notifications</description>
    <replay-support>false</replay-support>
    <access>
      <encoding>xml</encoding>
      <location>https://example.com/streams/SNMP</location>
    </access>
  </stream>
  <stream>
    <name>syslog-critical</name>
    <description>Critical and higher severity</description>
    <replay-support>true</replay-support>
    <replay-log-creation-time>
      2007-07-01T00:00:00Z
    </replay-log-creation-time>
    <access>
      <encoding>xml</encoding>
      <location>\
        https://example.com/streams/syslog-critical\
      </location>
    </access>
  </stream>
</streams>
```

### 6.3. Subscribing to Receive Notifications

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

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

The server will treat the connection as an event stream, using the Server Sent Events [W3C.REC-eventsource-20150203] transport strategy.

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

For example:

The client might send the following request:

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

The server might send the following response:

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

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

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

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

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



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

#### 6.3.1. NETCONF Event Stream

The server SHOULD support the "NETCONF" event stream defined in section 3.2.3 of [RFC5277]. The notification messages for this stream are encoded in XML.

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

The server MAY support the "start-time", "stop-time", and "filter" query parameters, defined in Section 4.8. Refer to Appendix D.3.6 for filter parameter examples.

#### 6.4. Receiving Event Notifications

RESTCONF notifications are encoded according to the definition of the event stream.

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

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

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

Two child nodes within the "notification" container are expected, representing the event time and the event payload. The "eventTime" node is defined within the same XML namespace as the "notification" element. It is defined to be within the "ietf-restconf" module namespace for JSON encoding purposes.

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

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

```
module example-mod {  
  namespace "http://example.com/event/1.0";  
  prefix ex;  
  
  notification event {  
    leaf event-class { type string; }  
    container reporting-entity {  
      leaf card { type string; }  
    }  
    leaf severity { type string; }  
  }  
}
```

An example SSE event notification encoded using XML:

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

An example SSE event notification encoded using JSON:

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

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

For example:

XML:

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

JSON:

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

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

## 7. Error Reporting

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

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

error-tag	status code
in-use	409
invalid-value	400, 404 or 406
(request) too-big	413
(response) too-big	400
missing-attribute	400
bad-attribute	400
unknown-attribute	400
bad-element	400
unknown-element	400
unknown-namespace	400
access-denied	401, 403
lock-denied	409
resource-denied	409
rollback-failed	500
data-exists	409
data-missing	409
operation-not-supported	405 or 501
operation-failed	412 or 500
partial-operation	500
malformed-message	400

Mapping from error-tag to status code

### 7.1. Error Response Message

When an error occurs for a request message on any resource type, and the status code that will be returned is in the "4xx" range (except for status code "403 Forbidden"), then the server **SHOULD** send a response message-body containing the information described by the "yang-errors" YANG data template within the "ietf-restconf" module, found in Section 8. The Content-Type of this response message **MUST** be "application/yang-data", plus optionally a structured syntax name suffix.

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

YANG Tree Diagram for <errors> data:

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

The semantics and syntax for RESTCONF error messages are defined with the "yang-errors" YANG data template extension, found in Section 8.

Examples:

The following example shows an error returned for an "lock-denied" error that can occur if a NETCONF client has locked a datastore. The RESTCONF client is attempting to delete a data resource. Note that an Accept header field is used to specify the desired encoding for the error message. There would be no response message-body content if this operation was successful.

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

The server might respond:

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

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

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

The client might send:

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

The server might respond:

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

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

## 8. RESTCONF Module

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

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

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

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

module ietf-restconf {
```

```
yang-version 1.1;
namespace "urn:ietf:params:xml:ns:yang:ietf-restconf";
prefix "rc";

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

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

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

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

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

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

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

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

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

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

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

// RFC Ed.: remove this note
```

```
// Note: extracted from draft-ietf-netconf-restconf-17.txt

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

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

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

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

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

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

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



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

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

The following data-def-stmt sub-statements are constrained when used within a yang-data-resource extension statement.

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

```
    ";
  }

  rc:yang-data yang-errors {
    uses errors;
  }

  rc:yang-data yang-api {
    uses restconf;
  }

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

    container errors {
      description
        "Represents an error report returned by the server if
         a request results in an error.";
```

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

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

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

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

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

  leaf error-message {
```

```
        type string;
        description
            "A message describing the error.";
    }

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

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

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

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

        container operations {
            description
                "Container for all operation resources.

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

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

```

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

```

To invoke the 'system-restart' RPC operation:

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

To discover the RPC operations supported by the server:

```
GET /restconf/operations
```

In XML the YANG module namespace identifies the module:

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

```

In JSON the YANG module name identifies the module:

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

```

";
}

leaf yang-library-version {
  type string {
    pattern '\d{4}-\d{2}-\d{2}';
  }
  config false;
  mandatory true;
  description
    "Identifies the revision date of the ietf-yang-library
    module that is implemented by this RESTCONF server.
    Indicates the year, month, and day in YYYY-MM-DD
    numeric format.";
}
}
}
}

```

<CODE ENDS>

## 9. RESTCONF Monitoring

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

YANG tree diagram for "ietf-restconf-monitoring" module:

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

### 9.1. restconf-state/capabilities

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

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

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

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

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

### 9.1.1.1. Query Parameter URIs

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

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

Name	Section	URI
depth	4.8.2	urn:ietf:params:restconf:capability:depth:1.0
fields	4.8.3	urn:ietf:params:restconf:capability:fields:1.0
filter	4.8.4	urn:ietf:params:restconf:capability:filter:1.0
replay	4.8.7	urn:ietf:params:restconf:capability:replay:1.0
with-defaults	4.8.9	urn:ietf:params:restconf:capability:with-defaults:1.0

RESTCONF Query Parameter URIs

### 9.1.1.2. The "defaults" Protocol Capability URI

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

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

RESTCONF defaults capability URI

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

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

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

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

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

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

Example: (split for display purposes only)

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

## 9.2. restconf-state/streams

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

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

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

## 9.3. RESTCONF Monitoring Module

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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



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

// RFC Ed.: remove this note
// Note: extracted from draft-ietf-netconf-restconf-17.txt

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

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

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

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

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

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

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

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

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

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

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

  leaf encoding {
    type string;
    description
      "This is the secondary encoding format within the
       'text/event-stream' encoding used by all streams.
       The type 'xml' is supported for XML encoding.
```

```
        The type 'json' is supported for JSON encoding.";
    }

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

<CODE ENDS>
```

## 10. YANG Module Library

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

### 10.1. modules-state/module

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

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

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

## 11. IANA Considerations

### 11.1. The "restconf" Relation Type

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

Relation Name: restconf

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

Reference: RFCXXXX

,

### 11.2. YANG Module Registry

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

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

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

This document registers two YANG modules in the YANG Module Names registry [RFC6020]:

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

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

### 11.3. Media Types

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

Type name: application

Subtype name: yang-data+xml

Required parameters: None

Optional parameters: None

Encoding considerations: 8-bit

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

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

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

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

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

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

Published specification: RFC XXXX

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

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

accessible as resources using the path in the request URI.

Additional information:

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

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

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

Intended usage: COMMON

Restrictions on usage: N/A

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

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

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

Provisional registration? (standards tree only): no

#### 11.3.2. Media Type application/yang-data+json

Type name: application

Subtype name: yang-data+json

Required parameters: None

Optional parameters: None

Encoding considerations: 8-bit  
Each conceptual YANG data node is encoded according to  
[RFC7951]. A data annotation is encoded according to  
[RFC7952].

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

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

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

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

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

Published specification: RFC XXXX

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

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

Additional information:

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

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

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

Intended usage: COMMON

Restrictions on usage: N/A

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

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

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

Provisional registration? (standards tree only): no

#### 11.4. RESTCONF Capability URNs

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

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

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

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

##### Index

##### Capability Identifier

-----

##### :defaults

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

##### :depth

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

##### :fields

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

##### :filter

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

##### :replay

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

##### :with-defaults

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



### 11.5. Registration of "restconf" URN sub-namespace

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

Registry Name: restconf

Specification: RFC XXXX

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

Repository: RESTCONF Capability URNs registry (Section 11.4)

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

## 12. Security Considerations

Section 2.1 states "A RESTCONF server MUST support the TLS protocol [RFC5246]". This language leaves open the possibility that a RESTCONF server might also support future versions of the TLS protocol. Of specific concern, TLS 1.3 [I-D.ietf-tls-tls13] introduces support for 0-RTT handshakes that can lead to security issues for REST APIs, as described in the Appendix of the TLS 1.3 specification. It is therefore RECOMMENDED that RESTCONF servers do not support 0-RTT at all (not even for idempotent requests) until an update to this RFC guides otherwise.

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

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

The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246]. The RESTCONF protocol uses the NETCONF access control model [RFC6536], which provides the means to

restrict access for particular RESTCONF users to a preconfigured subset of all available RESTCONF protocol operations and content.

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

Configuration information is by its very nature sensitive. Its transmission in the clear and without integrity checking leaves devices open to classic eavesdropping and false data injection attacks. Configuration information often contains passwords, user names, service descriptions, and topological information, all of which are sensitive. There are many patterns of attack that have been observed through operational practice with existing management interfaces. It would be wise for implementers to research them, and take them into account when implementing this protocol.

Different environments may well allow different rights prior to and then after authentication. When a RESTCONF operation is not properly authorized, the RESTCONF server MUST return a "401 Unauthorized" status-line. Note that authorization information can be exchanged in the form of configuration information, which is all the more reason to ensure the security of the connection. Note that it is possible for a client to detect configuration changes in data resources it is not authorized to access by monitoring changes in the ETag and Last-Modified header fields returned by the server for the datastore resource.

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

### 13. Acknowledgements

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

The authors would like to thank the following people for their excellent technical reviews of this document: Mehmet Ersue, Mahesh Jethanandani, Qin Wu, Joe Clarke, Bert Wijnen, Ladislav Lhotka, Rodney Cummings, Frank Xialiang, Tom Petch, Robert Sparks, Balint

Uveges, Randy Presuhn, Sue Hares, Mark Nottingham, Benoit Claise, Dale Worley, and Lionel Morand.

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

## 14. References

### 14.1. Normative References

- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3553] Mealling, M., Masinter, L., Hardie, T., and G. Klyne, "An IETF URN Sub-namespace for Registered Protocol Parameters", BCP 73, RFC 3553, DOI 10.17487/RFC3553, June 2003, <<http://www.rfc-editor.org/info/rfc3553>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, July 2008.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and T. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, March 2010.

- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, June 2011.
- [RFC6415] Hammer-Lahav, E. and B. Cook, "Web Host Metadata", RFC 6415, October 2011.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.
- [RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<http://www.rfc-editor.org/info/rfc7159>>.
- [RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.
- [RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.
- [RFC7232] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, June 2014.
- [RFC7235] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, June 2014.

- [RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, July 2014.
- [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <<http://www.rfc-editor.org/info/rfc7525>>.
- [RFC7589] Badra, M., Luchuk, A., and J. Schoenwaelder, "Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication", RFC 7589, DOI 10.17487/RFC7589, June 2015, <<http://www.rfc-editor.org/info/rfc7589>>.
- [RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, June 2016.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<http://www.rfc-editor.org/info/rfc7951>>.
- [RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <<http://www.rfc-editor.org/info/rfc7952>>.
- [W3C.REC-eventsource-20150203] Hickson, I., "Server-Sent Events", World Wide Web Consortium Recommendation REC-eventsource-20150203, February 2015, <<http://www.w3.org/TR/2015/REC-eventsource-20150203>>.
- [W3C.REC-xml-20081126] Yergeau, F., Maler, E., Paoli, J., Sperberg-McQueen, C., and T. Bray, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <<http://www.w3.org/TR/2008/REC-xml-20081126>>.
- [XPath] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999, <<http://www.w3.org/TR/1999/REC-xpath-19991116>>.

## 14.2. Informative References

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

Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", draft-ietf-netconf-yang-patch-12 (work in progress), October 2016.

[I-D.ietf-tls-tls13]

Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-18 (work in progress), October 2016.

[rest-dissertation]

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

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

## Appendix A. Change Log

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

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

### A.1. v17 to v18

- o addressed IESG review comments and clarifications
- o addressed Alexey's DISCUSS items
- o made Cache-Control MUST support, not SHOULD support
- o add example for PUT on a datastore
- o add IANA section for "restconf" URN sub-namespace
- o clarify media type file extensions

### A.2. v16 to v17

- o various clarifications from NETCONF WG mailing list
- o updated YANG 1.1 and YANG/JSON references to RFC numbers
- o fixed notification namespace and eventTime name bug

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

#### A.3. v15 to v16

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

#### A.4. v14 to v15

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

- o added clarifications from Gen-Art review by Robert Sparks
- o clarified terminology in many places

#### A.5. v13 - v14

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

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



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

## A.6. v12 - v13

- o fix YANG library module examples (now called module-state)
- o fix terminology idnit issue
- o removed RFC 2818 reference (changed citation to RFC 7230)

## A.7. v11 - v12

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

## A.8. v10 - v11

- o change term 'operational data' to 'state data'
- o clarify :startup behavior
- o clarify X.509 security text
- o change '403 Forbidden' to '401 Unauthorized' for GET error
- o clarify MUST have one "restconf" link relation
- o clarify that NV-storage is not mandatory

- o clarify how "Last-Modified" and "ETag" header info can be used by a client
- o clarify meaning of mandatory parameter
- o fix module name in action examples
- o clarify operation resource request needs to be known to parse the output
- o clarify ordered-by user terminology
- o fixed JSON example in D.1.1

#### A.9. v09 - v10

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

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

- o changed default value for content query parameter to "all"
- o changed empty container JSON encoding from "[null]" to "{}"
- o added mandatory /restconf/yang-library-version leaf to advertise revision-date of the YANG library implemented by the server
- o clarified URI encoding rules for leaf-list
- o clarified sec. 2.2 wrt/ certificates and TLS
- o added update procedure for entity tag and timestamp

#### A.10. v08 - v09

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

#### A.11. v07 - v08

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

## A.12. v06 - v07

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

## A.13. v05 - v06

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

## A.14. v04 - v05

- o changed term 'notification event' to 'event notification'
- o removed intro text about framework and meta-model
- o removed early mention of API resources
- o removed term unified datastore and cleaned up text about NETCONF datastores
- o removed text about not immediate persistence of edits
- o removed RESTCONF-specific data-resource-identifier typedef and its usage
- o clarified encoding of key leafs
- o changed several examples from JSON to XML encoding
- o made 'insert' and 'point' query parameters mandatory to implement
- o removed ":insert" capability URI
- o renamed stream/encoding to stream/access
- o renamed stream/encoding/type to stream/access/encoding
- o renamed stream/encoding/events to stream/access/location

- o changed XPath from informative to normative reference
- o changed rest-dissertation from normative to informative reference
- o changed example-jukebox playlist 'id' from a data-resource-identifier to a leafref pointing at the song name

#### A.15. v03 - v04

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

#### A.16. v02 - v03

- o added collection resource
- o added "page" query parameter capability

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

#### A.17. v01 - v02

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



- o closed "encoding of list keys" issue since no objection to new encoding of list keys in a target resource URI.
- o moved open issues list to the issue tracker on github

#### A.18. v00 - v01

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

## A.19. bierman:restconf-04 to ietf:restconf-00

- o updated open issues section

## Appendix B. Open Issues

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

The RESTCONF issues are tracked on github.com:

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

## Appendix C. Example YANG Module

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

YANG Tree Diagram for "example-jukebox" Module

```

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

```

rpcs:

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

#### C.1. example-jukebox YANG Module

```
module example-jukebox {

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

    organization "Example, Inc.";
    contact "support at example.com";
    description "Example Jukebox Data Model Module";
    revision "2016-08-15" {
        description "Initial version.";
        reference "example.com document 1-4673";
    }

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

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

```
}

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

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

  container library {

    description "Represents the jukebox library resource.";

    list artist {
      key name;

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

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

      list album {
        key name;

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

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

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

```
    }

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

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

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

    list song {
      key name;

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

      leaf name {
        type string {
          length "1 .. max";
        }
        description "The name of the song";
      }
      leaf location {
        type string;
        mandatory true;
        description
          "The file location string of the
          media file for the song";
      }
      leaf format {
        type string;
        description
          "An identifier string for the media type
          for the file associated with the
```

```
        'location' leaf for this entry.";
    }
    leaf length {
        type uint32;
        units "seconds";
        description
            "The duration of this song in seconds.";
    }
} // end list 'song'
} // end list 'album'
} // end list 'artist'

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

list playlist {
    key name;

    description
        "Example configuration data resource";

    leaf name {
        type string;
        description
            "The name of the playlist.";
    }
    leaf description {
        type string;
        description
            "A comment describing the playlist.";
    }
}
```

```
list song {
  key index;
  ordered-by user;

  description
    "Example nested configuration data resource";

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

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

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

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

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

## Appendix D. RESTCONF Message Examples

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

This section shows some typical RESTCONF message exchanges.

### D.1. Resource Retrieval Examples

#### D.1.1. Retrieve the Top-level API Resource

The client starts by retrieving the RESTCONF root resource:

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

The server might respond:

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

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

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

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

The server might respond as follows:



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

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

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

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

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

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

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

#### D.1.2. Retrieve The Server Module Information

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

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

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

The server might respond as follows:

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

{
  "ietf-yang-library:modules-state" : {
    "module-set-id" : "5479120c17a619545ea6aff7aa19838b036ebbd7",
    "module" : [
      {
        "name" : "foo",
        "revision" : "2012-01-02",
        "schema" : "https://example.com/modules/foo/2012-01-02",
        "namespace" : "http://example.com/ns/foo",
        "feature" : [ "feature1", "feature2" ],
        "deviation" : [
          {
            "name" : "foo-dev",
            "revision" : "2012-02-16"
          }
        ]
      },
      {
        "name" : "ietf-yang-library",
        "revision" : "2016-06-21",
        "schema" : "https://example.com/modules/\
ietf-yang-library/2016-06-21",
        "namespace" :
          "urn:ietf:params:xml:ns:yang:ietf-yang-library",
        "conformance-type" : "implement"
      }
    ],
    "conformance-type" : "implement"
  },
  {
    "name" : "foo-types",
    "revision" : "2012-01-05",
    "schema" :
      "https://example.com/modules/foo-types/2012-01-05",
    "namespace" : "http://example.com/ns/foo-types",
    "conformance-type" : "import"
  },
  {
    "name" : "bar",
    "revision" : "2012-11-05",
    "schema" : "https://example.com/modules/bar/2012-11-05",
```

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

#### D.1.3. Retrieve The Server Capability Information

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

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

The server might respond as follows:

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

```
<capabilities
  xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring">
  <capability>\
    urn:ietf:params:restconf:capability:defaults:1.0?\
      basic-mode=explicit\
  </capability>
  <capability>\
    urn:ietf:params:restconf:capability:with-defaults:1.0\
  </capability>
  <capability>\
    urn:ietf:params:restconf:capability:depth:1.0\
  </capability>
  <capability>\
    urn:ietf:params:restconf:capability:fields:1.0\
  </capability>
  <capability>\
    urn:ietf:params:restconf:capability:filter:1.0\
  </capability>
  <capability>\
    urn:ietf:params:restconf:capability:start-time:1.0\
  </capability>
  <capability>\
    urn:ietf:params:restconf:capability:stop-time:1.0\
  </capability>
  <capability>\
    http://example.com/capabilities/myparam\
  </capability>
</capabilities>
```

## D.2. Edit Resource Examples

### D.2.1. Create New Data Resources

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

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

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

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

```
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2016 17:02:00 GMT
Server: example-server
Location: https://example.com/restconf/data/\
    example-jukebox:jukebox/library/artist=Foo%20Fighters
Last-Modified: Mon, 23 Apr 2016 17:02:00 GMT
ETag: "b3830f23a4c"
```

To create a new "album" resource for this artist within the "jukebox" resource, the client might send the following request:

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

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

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

```
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2016 17:03:00 GMT
Server: example-server
Location: https://example.com/restconf/data/\
    example-jukebox:jukebox/library/artist=Foo%20Fighters/\
    album=Wasting%20Light
Last-Modified: Mon, 23 Apr 2016 17:03:00 GMT
ETag: "b8389233a4c"
```

#### D.2.2. Detect Resource Entity-Tag Change

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

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

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

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

```
HTTP/1.1 412 Precondition Failed
Date: Mon, 23 Apr 2016 19:01:00 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2016 17:45:00 GMT
ETag: "b34aed893a4c"
```

#### D.2.3. Edit a Datastore Resource

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

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

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

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

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

#### D.2.4. Replace a Datastore Resource

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

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

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

#### D.2.5. Edit a Data Resource

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

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

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

#### D.3. Query Parameter Examples

##### D.3.1. "content" Parameter

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



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

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

Example 1: content=all

To retrieve all the child resources, the "content" parameter is set to "all", or omitted, since this is the default value. The client might send:

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

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2016 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Content-Type: application/yang-data+json
```

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

#### Example 2: content=config

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

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

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2016 17:11:30 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2016 13:01:20 GMT
ETag: "eeeada438af"
Cache-Control: no-cache
Content-Type: application/yang-data+json

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

#### Example 3: content=nonconfig

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

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

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2016 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Content-Type: application/yang-data+json
```

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

#### D.3.2. "depth" Parameter

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

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

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

Example 1: depth=unbounded

To retrieve all the child resources, the "depth" parameter is not present or set to the default value "unbounded".

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

The server might respond:

```
HTTP/1.1 200 OK
```

Date: Mon, 23 Apr 2016 17:11:30 GMT  
Server: example-server  
Cache-Control: no-cache  
Content-Type: application/yang-data+json

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

```

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

```

Example 2: depth=1

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

```

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

```

The server might respond:

```

HTTP/1.1 200 OK
Date: Mon, 23 Apr 2016 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Content-Type: application/yang-data+json

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

```

Example 3: depth=3

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

```

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

```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2016 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Content-Type: application/yang-data+json
```

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

#### D.3.3. "fields" Parameter

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

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

The server might respond as follows.

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

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

```
{
  "ietf-restconf:data" : {
    "ietf-yang-library:modules-state" : {
      "module" : [
        {
          "name" : "example-jukebox",
          "revision" : "2015-06-04"
        },
        {
          "name" : "ietf-inet-types",
          "revision" : "2013-07-15"
        },
        {
          "name" : "ietf-restconf-monitoring",
          "revision" : "2016-03-16"
        },
        {
          "name" : "ietf-yang-library",
          "revision" : "2016-06-21"
        },
        {
          "name" : "ietf-yang-types",
          "revision" : "2013-07-15"
        }
      ]
    }
  }
}
```

#### D.3.4. "insert" Parameter

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

Request from client:



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

{
  "example-jukebox:song" : [
    {
      "index" : 1,
      "id" : "/example-jukebox:jukebox/library\
        /artist[name='Foo Figthers']\
        /album[name='Wasting Light']\
        /song[name='Rope']"
    }
  ]
}
```

Response from server:

```
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2016 13:01:20 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2016 13:01:20 GMT
Location: https://example.com/restconf/data/\
    example-jukebox:jukebox/playlist=Foo-One/song=1
ETag: "eeeada438af"
```

#### D.3.5. "point" Parameter

In this example, the client is inserting a new song entry in the "Foo-One" playlist after the first song.

Request from client:

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

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

Response from server:

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

#### D.3.6. "filter" Parameter

The following URIs show some examples of notification filter specifications:

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

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

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

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

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

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

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

#### D.3.7. "start-time" Parameter

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

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

#### D.3.8. "stop-time" Parameter

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

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

## D.3.9. "with-defaults" Parameter

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

If the server defaults-uri basic-mode is "trim", the the following request for interface "eth1" might be as follows:

Without query parameter:

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

The server might respond as follows.

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

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

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

With query parameter:

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

The server might respond as follows.

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

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

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

#### Authors' Addresses

Andy Bierman  
YumaWorks

Email: andy@yumaworks.com

Martin Bjorklund  
Tail-f Systems

Email: mbj@tail-f.com

Kent Watsen  
Juniper Networks

Email: kwatsen@juniper.net

NETCONF Working Group  
Internet-Draft  
Updates: 4253 (if approved)  
Intended status: Standards Track  
Expires: January 02, 2015

K. Watsen  
Juniper Networks  
July 2014

NETCONF Call Home using SSH  
draft-ietf-netconf-reverse-ssh-06

Abstract

This document presents a technique for a NETCONF server to request that a NETCONF client initiates a SSH connection to the NETCONF server, a technique referred to as 'call home'. Call home is needed to support deployments where the NETCONF client is otherwise unable to initiate a SSH connection to the NETCONF server directly.

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 January 02, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1. Requirements Terminology . . . . .	2
2. Introduction . . . . .	2
2.1. Applicability Statement . . . . .	3
2.2. Update to RFC 4253 . . . . .	3
2.3. Draft Naming . . . . .	3
3. Benefits to Device Management . . . . .	3
4. Protocol . . . . .	5
5. SSH Server Identification and Verification . . . . .	5
6. Device Configuration . . . . .	6
7. Security Considerations . . . . .	7
8. IANA Considerations . . . . .	8
9. Acknowledgements . . . . .	8
10. References . . . . .	8
10.1. Normative References . . . . .	8
10.2. Informative References . . . . .	9
Appendix A. Change Log . . . . .	9
A.1. 05 to 06 . . . . .	9
A.2. 04 to 05 . . . . .	10
A.3. 03 to 04 . . . . .	10
A.4. 02 to 03 . . . . .	11
A.5. 01 to 02 . . . . .	11
A.6. 00 to 01 . . . . .	11

## 1. Requirements 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].

## 2. Introduction

This document presents a technique for a NETCONF server to request that a NETCONF [RFC6241] client initiates a SSH [RFC4251] connection to the NETCONF server, a technique referred to as 'call home'. Call home is needed to support deployments where the NETCONF client is otherwise unable to initiate a SSH connection to the NETCONF server directly.

## 2.1. Applicability Statement

The techniques described in this document are suitable for network management scenarios such as the ones described in section 3. However, these techniques SHOULD only be used for a NETCONF server to initiate a connection to a NETCONF client, as described in this document.

The reason for this restriction is that different protocols have different security assumptions. The NETCONF over SSH specification requires NETCONF clients and servers to verify the identity of the other party before starting the NETCONF protocol (section 6 of [RFC6242]). This contrasts with the base SSH protocol, which does not require programmatic verification of the other party (section 9.3.4 of [RFC4251] and section 4 of [RFC4252]). In such circumstances, allowing the SSH server to contact the SSH client would open new vulnerabilities. Therefore, any use of call home with SSH for purposes other than NETCONF will need a thorough, contextual security analysis.

## 2.2. Update to RFC 4253

This document updates the SSH Transport Layer Protocol [RFC4253] only by removing the restriction in Section 4 (Connection Setup) of [RFC4252] that the SSH Client must initiate the transport connection. Security implications related to this change are discussed in Security Considerations (Section 7).

## 2.3. Draft Naming

(this section should be removed if this draft becomes an RFC)

This draft's name includes the string "reverse-ssh", and yet currently nowhere in this draft is there any reference to reversing SSH. This apparent omission comes from the -05 edit of this draft, where "Reverse SSH" was changed to "Call Home" throughout. If this draft becomes an RFC, its name would no longer contain the obsolete "reverse-ssh" reference, thus self-correcting this inconsistency.

## 3. Benefits to Device Management



The SSH protocol is nearly ubiquitous for device management, as it is the transport for the command-line applications 'ssh', 'scp', and 'sftp' and is the required transport for the NETCONF protocol [RFC6241]. However, all these SSH-based protocols expect the network element to be the SSH server.

NETCONF over SSH Call Home enables the network element to consistently be the SSH server regardless of which peer initiates the underlying TCP connection. Maintaining the role of SSH server is both necessary and desirable. It is necessary because SSH channels and subsystems can only be opened on the SSH server. It is desirable because it conveniently leverages infrastructure that may be deployed for host-key verification and user authentication.

Call home is useful for both initial deployment and on-going device management and may be used to enable any of the following scenarios:

- o The network element may proactively call home after being powered on for the first time to register itself with its management system.
- o The network element may access the network in a way that dynamically assigns it an IP address and it doesn't register its assigned IP address to a mapping service.
- o The network element may be configured in "stealth mode" and thus doesn't have any open ports for the management system to connect to.
- o The network element may be deployed behind a firewall that doesn't allow SSH access to the internal network.
- o The network element may be deployed behind a firewall that implements network address translation (NAT) for all internal network IP addresses, thus complicating the ability for a management system to connect to it.
- o The operator may prefer to have network elements initiate management connections believing it is easier to secure one open-port in the data center than to have an open port on each network element in the network.

One key benefit of using SSH as the transport protocol is its ability to multiplex an unspecified number of independently flow-controlled TCP sessions [RFC4254]. This is valuable as the network element only needs to be configured to initiate a single call home connection to a management system, regardless the number of NETCONF channels the management system wants to open.

#### 4. Protocol

The NETCONF server's perspective (e.g., the network element)

- o The NETCONF server initiates a TCP connection to the NETCONF client on the IANA-assigned SSH for NETCONF Call Home port YYYY.
- o The TCP connection is accepted and a TCP session is established.
- o Using this TCP connection, the NETCONF server immediately starts the SSH server protocol. That is, the next message sent on the TCP stream is SSH's Protocol Version Exchange message (section 4.2, [RFC4253]).
- o The SSH connection is established.

The NETCONF client's perspective (e.g., the management system)

- o The NETCONF client listens for TCP connections on the IANA-assigned NETCONF over SSH Call Home port YYYY.
- o The NETCONF client accepts an incoming TCP connection and a TCP session is established.
- o Using this TCP connection, the NETCONF client immediately starts the SSH Client protocol, starting with sending the SSH's Protocol Version Exchange message (section 4.2, [RFC4253]).
- o The SSH connection is established.

#### 5. SSH Server Identification and Verification

When the management system accepts a new incoming TCP connection on the NETCONF over SSH Call Home port, it starts the SSH client protocol. As the SSH client, it MUST authenticate the SSH server, by both identifying the network element and verifying its SSH host key.

Due to call home having the network element initiate the TCP connection, the management system MAY identify the remote peer using the source IP address of the TCP connection. However, identifying the remote peer using the source IP address of the TCP connection is NOT RECOMMENDED as it can only work in networks that use known static addresses.

To support network elements having dynamically-assigned IP addresses, or deployed behind gateways that translate their IP addresses (e.g., NAT), the management system MAY identify the device using its SSH host key. For instance, a fingerprint of the network element's host

key could itself be used as an identifier since each device has a statistically unique host key. However, identifying the remote peer using its host key directly is NOT RECOMMENDED as it requires the host key to be manually verified the first time the network element connects and anytime its host key changes thereafter.

Yet another option for identifying the network element is for its host key to encode the network element's identity, such as if the host key were a certificate. This option enables the host key to change over time, so long as it continues to encode the same identity, but brings the next issue of how the management system can verify the network element's host key is authentic.

The security of SSH is anchored in the ability for the SSH client to verify the SSH server's host key. Typically this is done by comparing the host key presented by the SSH server with one that was previously configured on the SSH client, looking it up in a local database using the identity of the SSH client as the lookup key. Nothing changes regarding this requirement due to the direction reversal of the underlying TCP connection. To ensure security, the management system MUST verify the network element's SSH host key each time a SSH session is established.

However, configuring distinct host keys on the management system doesn't scale well, which is an important consideration to a network management system. A more scalable strategy for the management system is for the network element's manufacturer to sign the network-element's host key with a common trusted key, such as a certificate authority. Then, when the network-element is deployed, the management system only needs to trust a single certificate, which vouches for the authenticity of the various network element host keys.

Since both the identification and verification issues are addressed using certificates, this draft RECOMMENDS network elements use a host key that can encode a unique identifier (e.g., its serial number) and be signed by a common trust anchor (e.g., a certificate authority). Examples of suitable public host keys are the X.509v3 keys defined in defined in [RFC6187] and the PGP keys defined in [RFC4253].

## 6. Device Configuration

How to configure a device to initiate a NETCONF over SSH Call Home connection is outside the scope of this document, as implementations can support this protocol using a proprietary configuration data model. That said, a YANG [RFC6020] model to configure NETCONF over SSH Call Home is specified in [draft-ietf-netconf-server-model].

## 7. Security Considerations

This RFC deviates from standard SSH protocol usage by allowing the SSH server to initiate the TCP connection. This conflicts with section 4 of the SSH Transport Layer Protocol RFC [RFC4253], which states "The client initiates the connection". However this statement is made without rationalization and it's not clear how it impacts the security of the protocol, so this section analyzes the security offered by having the client initiate the connection.

First, assuming the SSH server is not using a public host key algorithm that certifies its identity, the security of the protocol doesn't seem to be sensitive to which peer initiates the connection. That is, it is still the case that reliable distribution of host keys (or their fingerprints) should occur prior to first connection and that verification for subsequent connections happens by comparing the host keys in a locally cached database. It does not seem to matter if the SSH server's host name is derived from user-input or extracted from the TCP layer, potentially via a reverse-DNS lookup. Once the host name-to-key association is stored in a local database, no man-in-the-middle attack is possible due to the attacker being unable to guess the real SSH server's private key (Section 9.3.4 (Man-in-the-middle) of [RFC4251]).

That said, this RFC recommends implementations use a public host key algorithm that certifies the SSH server's identity. The identity can be any unique identifier, such as a device's serial number or a deployment-specific value. If this recommendation is followed, then no information from the TCP layer would be needed to lookup the device in a local database and therefore the directionality of the TCP layer is clearly inconsequential.

The SSH protocol negotiates which algorithms it will use during key exchange (Section 7.1 (Algorithm Negotiation) in [RFC4253]). The algorithm selected is essentially the first compatible algorithm listed by the SSH client that is also listed by the SSH server. For a network management application, there may be a need to advertise a large number of algorithms to be compatible with the various devices it manages. The SSH client SHOULD order its list of public host key algorithms such that all the certifiable public host key algorithms are listed first. Additionally, when possible, SSH servers SHOULD only list certifiable public host key algorithms. Note that since the SSH server would have to be configured to know which IP address it is to connect to, it is expected that it will also be configured to know which host key algorithm to use for the particular application, and hence only needs to list just that one public host key algorithm.

This RFC suggests implementations can use a device's serial number as a form of identity. A potential concern with using a serial number is that the SSH protocol passes the SSH server's host-key in the clear and many times serial numbers encode revealing information about the device, such as what kind of device it is and when it was manufactured. While there is little security in trying to hide this information from an attacker, it is understood that some deployments may want to keep this information private. If this is a concern, deployments SHOULD use an alternate unique identifier, if even just the hash of the device's serial number.

An attacker could DoS the application by having it perform computationally expensive operations, before deducing that the attacker doesn't possess a valid key. This is no different than any secured service and all common precautions apply (e.g., blacklisting the source address after a set number of unsuccessful login attempts).

## 8. IANA Considerations

This document requests that IANA assigns a TCP port number in the "Registered Port Numbers" range with the service name "netconf-ssh-ch". This port will be the default port for NETCONF over SSH Call Home protocol and will be used when the NETCONF server is to initiate a connection to a NETCONF client using SSH. Below is the registration template following the rules in [RFC6335].

Service Name:	netconf-ssh-ch
Transport Protocol(s):	TCP
Assignee:	IESG <iesg@ietf.org>
Contact:	IETF Chair <chair@ietf.org>
Description:	NETCONF over SSH Call Home
Reference:	RFC XXXX
Port Number:	YYYY

## 9. Acknowledgements

The author would like to thank for following for lively discussions on list and in the halls (ordered by last name): Andy Bierman, Martin Bjorklund, Mehmet Ersue, Wes Hardaker, Stephen Hanna, David Harrington, Jeffrey Hutzelman, Radek Krejci, Alan Luchuk, Mouse, Russ Mundy, Tom Petch, Peter Saint-Andre, Joe Touch, Sean Turner, Bert Wijnen.

## 10. References

### 10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels ", BCP 14, RFC 2119, March 1997.
- [RFC4250] Lehtinen, S. and C. Lonvick, "The Secure Shell (SSH) Protocol Assigned Numbers ", RFC 4250, December 2005.
- [RFC4251] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture ", RFC 4251, January 2006.
- [RFC4252] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Authentication Protocol ", RFC 4252, January 2006.
- [RFC4253] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Transport Layer Protocol ", RFC 4253, January 2006.
- [RFC4254] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Connection Protocol ", RFC 4254, January 2006.
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF) ", RFC 6020, October 2010.
- [RFC6187] Igoe, K. and D. Stebila, "X.509v3 Certificates for Secure Shell Authentication ", RFC 6187, March 2011.
- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "NETCONF Configuration Protocol", RFC 6241, June 2011.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", RFC 6335, August 2011.

## 10.2. Informative References

- [draft-ietf-netconf-server-model]  
Watsen, K. and J. Schoenwaelder, "A YANG Data Model for NETCONF Server Configuration", RFC 6242, June 2011.

## Appendix A. Change Log

### A.1. 05 to 06

Changed title to "NETCONF Call Home using SSH"

Revised the Abstract and Introduction to better explain what the document regards.

Changed "MUST" to "SHOULD" in the Applicability Statement.

Added a "Draft Naming" section explaining why, despite its name, reversing SSH is nowhere in the text

Added PGP keys as another kind of SSH host key encoding identity and signed by a trust anchor.

Revised the Device Considerations section to more clearly explain why a device configuration data model is out of scope, and hence an Informative reference.

Clarified Security Considerations section on use of serial numbers.

A.2. 04 to 05

Changed "Reverse SSH" to "Call Home"

Added references to Applicability Statement

A.3. 03 to 04

Changed title to "Reverse SSH for NETCONF Call Home" (changed again in -05)

Removed statement on how other SSH channels might be used for other protocols

Improved language on how the management system, as the SSH client, MUST authenticate the SSH server

Clarified that identifying the network element using source IP address is NOT RECOMMENDED

Clarified that identifying the NE using simple certificate comparison is NOT RECOMMENDED

Device Configuration section now more clearly states that the YANG model is out of scope

Change requested port name to "netconf-ssh-ch"

General edits for grammar, capitalization, and spellings

A.4. 02 to 03

Updated Device Configuration section to reference  
[draft-ietf-netconf-server-model]

A.5. 01 to 02

Added Applicability Statement

Removed references to ZeroConf / ZeroTouch

Clarified the protocol section

Added a section for identification and verification

A.6. 00 to 01

Removed the hmac-\* family of algorithms

Author's Address

Kent Watsen  
Juniper Networks

EMail: kwatsen@juniper.net



NETCONF Working Group  
Internet-Draft  
Obsoletes: 5539 (if approved)  
Intended status: Standards Track  
Expires: October 12, 2015

M. Badra  
Zayed University  
A. Luchuk  
SNMP Research, Inc.  
J. Schoenwaelder  
Jacobs University Bremen  
April 10, 2015

Using the NETCONF Protocol over Transport Layer Security (TLS) with  
Mutual X.509 Authentication  
draft-ietf-netconf-rfc5539bis-10

## Abstract

The Network Configuration Protocol (NETCONF) provides mechanisms to install, manipulate, and delete the configuration of network devices. This document describes how to use the Transport Layer Security (TLS) protocol with mutual X.509 authentication to secure the exchange of NETCONF messages. This revision of RFC 5539 documents the new message framing used by NETCONF 1.1 and it obsoletes RFC 5539.

## 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 October 12, 2015.

## Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Connection Initiation . . . . .	3
3. Message Framing . . . . .	3
4. Connection Closure . . . . .	3
5. Certificate Validation . . . . .	3
6. Server Identity . . . . .	4
7. Client Identity . . . . .	4
8. Cipher Suites . . . . .	6
9. Security Considerations . . . . .	6
10. IANA Considerations . . . . .	7
11. Acknowledgements . . . . .	7
12. References . . . . .	8
12.1. Normative References . . . . .	8
12.2. Informative References . . . . .	8
Appendix A. Changes from RFC 5539 . . . . .	9
Authors' Addresses . . . . .	9

## 1. Introduction

The NETCONF protocol [RFC6241] defines a mechanism through which a network device can be managed. NETCONF is connection-oriented, requiring a persistent connection between peers. This connection must provide integrity, confidentiality, peer authentication, and reliable, sequenced data delivery.

This document defines how NETCONF messages can be exchanged over Transport Layer Security (TLS) [RFC5246]. Implementations MUST support mutual TLS certificate-based authentication [RFC5246]. This assures the NETCONF server of the identity of the principal who wishes to manipulate the management information. It also assures the NETCONF client of the identity of the server for which it wishes to manipulate the management information.

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

## 2. Connection Initiation

The peer acting as the NETCONF client MUST act as the TLS client. The TLS client actively opens the TLS connection and the TLS server passively listens for the incoming TLS connections. The well-known TCP port number 6513 is used by NETCONF servers to listen for TCP connections established by NETCONF over TLS clients. The TLS client MUST send the TLS ClientHello message to begin the TLS handshake. The TLS server MUST send a CertificateRequest in order to request a certificate from the TLS client. Once the TLS handshake has finished, the client and the server MAY begin to exchange NETCONF messages. Client and server identity verification is done before the NETCONF <hello> message is sent. This means that the identity verification is completed before the NETCONF session is started.

## 3. Message Framing

All NETCONF messages MUST be sent as TLS "application data". It is possible that multiple NETCONF messages be contained in one TLS record, or that a NETCONF message be transferred in multiple TLS records.

The previous version of this document [RFC5539] used the framing sequence defined in [RFC4742]. This version aligns with [RFC6242] and adopts the framing protocol defined in [RFC6242] as follows:

The NETCONF <hello> message MUST be followed by the character sequence ]]>]]. Upon reception of the <hello> message, the peers inspect the announced capabilities. If the :base:1.1 capability is advertised by both peers, the chunked framing mechanism defined in Section 4.2 of [RFC6242] is used for the remainder of the NETCONF session. Otherwise, the old end-of-message-based mechanism (see Section 4.3 of [RFC6242]) is used.

## 4. Connection Closure

A NETCONF server will process NETCONF messages from the NETCONF client in the order in which they are received. A NETCONF session is closed using the <close-session> operation. When the NETCONF server processes a <close-session> operation, the NETCONF server SHALL respond and close the TLS session as described in Section 7.2.1 of [RFC5246].

## 5. Certificate Validation

Both peers MUST use X.509 certificate path validation [RFC5280] to verify the integrity of the certificate presented by the peer. The presented X.509 certificate may also be considered valid if it

matches one obtained by another trusted mechanism, such as using a locally configured certificate fingerprint. If X.509 certificate path validation fails and the presented X.509 certificate does not match a certificate obtained by a trusted mechanism, the connection MUST be terminated as defined in [RFC5246].

## 6. Server Identity

The NETCONF client MUST check the identity of the server according to Section 6 of [RFC6125].

## 7. Client Identity

The NETCONF server MUST verify the identity of the NETCONF client to ensure that the incoming request to establish a NETCONF session is legitimate before the NETCONF session is started.

The NETCONF protocol [RFC6241] requires that the transport protocol's authentication process results in an authenticated NETCONF client identity whose permissions are known to the server. The authenticated identity of a client is commonly referred to as the NETCONF username. The following algorithm is used by the NETCONF server to derive a NETCONF username from a certificate. (Note that the algorithm below is the same as the one described in the SNMP-TLS-TM-MIB MIB module defined in [RFC6353] and in the ietf-x509-cert-to-name YANG module defined in [RFC7407].)

- (a) The server maintains an ordered list of mappings of certificates to NETCONF usernames. Each list entry contains
  - \* a certificate fingerprint (used for matching the presented certificate),
  - \* a map type (indicates how the NETCONF username is derived from the certificate), and
  - \* optional auxiliary data (used to carry a NETCONF username if the map type indicates the user name is explicitly configured).
- (b) The NETCONF username is derived by considering each list entry in order. The fingerprint member of the current list entry determines whether the current list entry is a match:
  1. If the list entry's fingerprint value matches the fingerprint of the presented certificate, then consider the list entry as a successful match.

2. If the list entry's fingerprint value matches that of a locally held copy of a trusted CA certificate, and that CA certificate was part of the CA certificate chain to the presented certificate, then consider the list entry as a successful match.
- (c) Once a matching list entry has been found, the map type of the current list entry is used to determine how the username associated with the certificate should be determined. Possible mapping options are:
- A. The username is taken from the auxiliary data of the current list entry. This means the username is explicitly configured (map type 'specified').
  - B. The subjectAltName's rfc822Name field is mapped to the username (map type 'san-rfc822-name'). The local part of the rfc822Name is used unaltered but the host-part of the name must be converted to lowercase.
  - C. The subjectAltName's dNSName is mapped to the username (map type 'san-dns-name'). The characters of the dNSName are converted to lowercase.
  - D. The subjectAltName's iPAddress is mapped to the username (map type 'san-ip-address'). IPv4 addresses are converted into decimal-dotted quad notation (e.g., '192.0.2.1'). IPv6 addresses are converted into a 32-character all lowercase hexadecimal string without any colon separators.
  - E. Any of the subjectAltName's rfc822Name, dNSName, iPAddress is mapped to the username (map type 'san-any'). The first matching subjectAltName value found in the certificate of the above types MUST be used when deriving the name.
  - F. The certificate's CommonName is mapped to the username (map type 'common-name'). The CommonName is converted to UTF-8 encoding. The usage of CommonNames is deprecated and users are encouraged to use subjectAltName mapping methods instead.
- (d) If it is impossible to determine a username from the list entry's data combined with the data presented in the certificate, then additional list entries MUST be searched looking for another potential match. Similarly, if the username does not comply to the NETCONF requirements on usernames [RFC6241], then additional list entries MUST be

searched looking for another potential match. If there are no further list entries, the TLS session MUST be terminated.

The username provided by the NETCONF over TLS implementation will be made available to the NETCONF message layer as the NETCONF username without modification.

The NETCONF server configuration data model [I-D.ietf-netconf-server-model] covers NETCONF over TLS and provides further details such as certificate fingerprint formats exposed to network configuration systems.

## 8. Cipher Suites

Implementations MUST support TLS 1.2 [RFC5246] and are REQUIRED to support the mandatory-to-implement cipher suite. Implementations MAY implement additional TLS cipher suites that provide mutual authentication [RFC5246] and confidentiality as required by NETCONF [RFC6241]. Implementations SHOULD follow the recommendations given in [I-D.ietf-uta-tls-bcp].

## 9. Security Considerations

NETCONF is used to access configuration and state information and to modify configuration information, so the ability to access this protocol should be limited to users and systems that are authorized to view the NETCONF server's configuration and state or to modify the NETCONF server's configuration.

Configuration or state data may include sensitive information, such as usernames or security keys. So, NETCONF requires communications channels that provide strong encryption for data privacy. This document defines a NETCONF over TLS mapping that provides for support of strong encryption and authentication. The security considerations for TLS [RFC5246] and NETCONF [RFC6241] apply here as well.

NETCONF over TLS requires mutual authentication. Neither side should establish a NETCONF over TLS connection with an unknown, unexpected, or incorrect identity on the opposite side. Note that the decision whether a certificate presented by the client is accepted can depend on whether a trusted CA certificate is white listed (see Section 7). If deployments make use of this option, it is recommended that the white listed CA certificate is used only to issue certificates that are used for accessing NETCONF servers. Should the CA certificate be used to issue certificates for other purposes, then all certificates created for other purposes will be accepted by a NETCONF server as well, which is likely not suitable.

This document does not support third-party authentication (e.g., backend Authentication, Authorization, and Accounting (AAA) servers) due to the fact that TLS does not specify this way of authentication and that NETCONF depends on the transport protocol for the authentication service. If third-party authentication is needed, the SSH transport [RFC6242] can be used.

RFC 5539 assumes that the end-of-message (EOM) sequence, `]]>]]>`, cannot appear in any well-formed XML document, which turned out to be mistaken. The EOM sequence can cause operational problems and open space for attacks if sent deliberately in NETCONF messages. It is however believed that the associated threat is not very high. This document still uses the EOM sequence for the initial `<hello>` message to avoid incompatibility with existing implementations. When both peers implement `:base:1.1` capability, a proper framing protocol (chunked framing mechanism; see Section 3) is used for the rest of the NETCONF session, to avoid injection attacks.

## 10. IANA Considerations

Based on the previous version of this document, RFC 5539, IANA has assigned a TCP port number (6513) in the "Registered Port Numbers" range with the service name "netconf-tls". This port will be the default port for NETCONF over TLS, as defined in Section 2. Below is the registration template following the rules in [RFC6335].

Service Name:	netconf-tls
Transport Protocol(s):	TCP
Assignee:	IESG <iesg@ietf.org>
Contact:	IETF Chair <chair@ietf.org>
Description:	NETCONF over TLS
Reference:	RFC XXXX
Port Number:	6513

[[CREF1: RFC Editor: Please replace XXXX above with the allocated RFC number and remove this comment. --JS]]

## 11. Acknowledgements

The authors like to acknowledge Martin Bjorklund, Olivier Coupelon, Mehmet Ersue, Stephen Farrell, Miao Fuyou, Ibrahim Hajjeh, David Harrington, Sam Hartman, Alfred Hoenes, Simon Josefsson, Barry Leiba, Tom Petch, Eric Rescorla, Dan Romascanu, Kent Watsen, Bert Wijnen, Stefan Winter and the NETCONF mailing list members for their comments on this document. Charlie Kaufman, Pasi Eronen, and Tim Polk provided a thorough review of previous versions of this document.

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

## 12. References

### 12.1. Normative References

- [I-D.ietf-uta-tls-bcp] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of TLS and DTLS", draft-ietf-uta-tls-bcp-09 (work in progress), February 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, March 2011.
- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, August 2011.

### 12.2. Informative References



- [I-D.ietf-netconf-server-model]  
Watsen, K. and J. Schoenwaelder, "NETCONF Server and RESTCONF Server Configuration Models", draft-ietf-netconf-server-model-06 (work in progress), February 2015.
- [RFC4742] Wasserman, M. and T. Goddard, "Using the NETCONF Configuration Protocol over Secure SHell (SSH)", RFC 4742, December 2006.
- [RFC5539] Badra, M., "NETCONF over Transport Layer Security (TLS)", RFC 5539, May 2009.
- [RFC6353] Hardaker, W., "Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP)", STD 78, RFC 6353, July 2011.
- [RFC7407] Bjorklund, M. and J. Schoenwaelder, "A YANG Data Model for SNMP Configuration", RFC 7407, December 2014.

#### Appendix A. Changes from RFC 5539

This section summarizes major changes between this document and RFC 5539.

- o Documented that NETCONF over TLS uses the new message framing if both peers support the :base:1.1 capability.
- o Removed redundant text that can be found in the TLS and NETCONF specifications and restructured the text. Alignment with [RFC6125].
- o Added a high-level description how NETCONF usernames are derived from certificates.
- o Removed the reference to BEEP.

#### Authors' Addresses

Mohamad Badra  
Zayed University

Email: mbadra@gmail.com

Alan Luchuk  
SNMP Research, Inc.  
3001 Kimberlin Heights Road  
Knoxville, TN 37920  
USA

Phone: +1 865 573 1434  
Email: [luchuk@snmp.com](mailto:luchuk@snmp.com)  
URI: <http://www.snmp.com/>

Juergen Schoenwaelder  
Jacobs University Bremen  
Campus Ring 1  
28759 Bremen  
Germany

Phone: +49 421 200 3587  
Email: [j.schoenwaelder@jacobs-university.de](mailto:j.schoenwaelder@jacobs-university.de)  
URI: <http://www.jacobs-university.de/>

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: May 26, 2017

A. Bierman  
YumaWorks  
M. Bjorklund  
Tail-f Systems  
K. Watsen  
Juniper Networks  
November 22, 2016

YANG Patch Media Type  
draft-ietf-netconf-yang-patch-14

Abstract

This document describes a method for applying patches to configuration 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 May 26, 2017.

Copyright Notice

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

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

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

## Table of Contents

1. Introduction . . . . .	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. Examples . . . . .	5
1.1.7. Tree Diagram Notations . . . . .	6
2. YANG Patch . . . . .	6
2.1. Target Resource . . . . .	7
2.2. yang-patch Request . . . . .	8
2.3. yang-patch-status Response . . . . .	9
2.4. Target Data Node . . . . .	10
2.5. Edit Operations . . . . .	11
2.6. Successful Edit Response Handling . . . . .	11
2.7. Error Handling . . . . .	11
2.8. yang-patch RESTCONF Capability . . . . .	12
3. YANG Module . . . . .	12
4. IANA Considerations . . . . .	21
4.1. YANG Module Registry . . . . .	21
4.2. Media Types . . . . .	21
4.2.1. Media Type application/yang-patch+xml . . . . .	21
4.2.2. Media Type application/yang-patch+json . . . . .	23
4.3. RESTCONF Capability URNs . . . . .	25
5. Security Considerations . . . . .	25
6. Normative References . . . . .	26
Appendix A. Acknowledgements . . . . .	27
Appendix B. Change Log . . . . .	27
B.1. v12 to v13 . . . . .	27
B.2. v11 to v12 . . . . .	27
B.3. v10 to v11 . . . . .	28
B.4. v09 to v10 . . . . .	28
B.5. v08 to v09 . . . . .	28
B.6. v07 to v08 . . . . .	29
B.7. v06 to v07 . . . . .	29
B.8. v05 to v06 . . . . .	29
B.9. v04 to v05 . . . . .	29
B.10. v03 to v04 . . . . .	30
B.11. v02 to v03 . . . . .	30
B.12. v01 to v02 . . . . .	30
B.13. v00 to v01 . . . . .	30
B.14. birman:yang-patch-00 to ietf:yang-patch-00 . . . . .	31

Appendix C. Open Issues . . . . .	31
Appendix D. Example YANG Module . . . . .	31
D.1. YANG Patch Examples . . . . .	32
D.1.1. Add Resources: Error . . . . .	32
D.1.2. Add Resources: Success . . . . .	36
D.1.3. Insert list entry example . . . . .	38
D.1.4. Move list entry example . . . . .	40
D.1.5. Edit datastore resource example . . . . .	41
Authors' Addresses . . . . .	43

## 1. Introduction

There is a need for standard mechanisms to patch datastores defined in [RFC6241], which contain conceptual data that conforms to schema specified with YANG [RFC7950]. An "ordered edit list" approach is needed to provide RESTCONF client developers with more precise RESTCONF client control of the edit procedure than existing mechanisms found in [I-D.ietf-netconf-restconf].

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]. This document only specifies the use of the YANG Patch media type with the RESTCONF protocol.

It may be possible to use YANG Patch with other protocols besides RESTCONF. This is outside the scope of this document. For any protocol which supports the YANG Patch media type, if the entire patch document cannot be successfully applied, then the server MUST NOT apply any of the changes. It may be possible to use YANG Patch with datastore types other than a configuration datastore. This is outside the scope of this document.

### 1.1. Terminology

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

#### 1.1.1. NETCONF

The following terms are defined in [RFC6241]:

- o configuration data
- o datastore
- o configuration datastore

- o protocol operation
- o running configuration datastore
- o state data
- o user

#### 1.1.2. HTTP

The following terms are defined in [RFC7230]:

- o header field
- o message-body
- o query
- o request URI

The following terms are defined in [RFC7231]:

- o method
- o request
- o resource

#### 1.1.3. YANG

The following terms are defined in [RFC7950]:

- o container
- o data node
- o leaf
- o leaf-list
- o list
- o RPC operation (now called protocol operation)

#### 1.1.4. RESTCONF

The following terms are defined in [I-D.ietf-netconf-restconf]:

- o application/yang-data+xml
- o application/yang-data+json
- o data resource
- o datastore resource
- o patch
- o RESTCONF capability
- o target resource
- o YANG data template

#### 1.1.5. YANG Patch

The following terms are used within this document:

- o RESTCONF client: a client which implements the RESTCONF protocol.
- o RESTCONF server: a server which implements the RESTCONF protocol.
- o YANG Patch: a conceptual edit request using the "yang-patch" YANG Patch template, defined in Section 3. In HTTP, refers to a PATCH method where a representation uses either the media type "application/yang-patch+xml" or "application/yang-patch+json".
- o YANG Patch Status: a conceptual edit status response using the YANG "yang-patch-status" YANG data template, defined in Section 3. In HTTP, refers to a response message for a PATCH method, where it has a representation with either the media type "application/yang-data+xml" or "application/yang-data+json".
- o YANG Patch template: this is similar to a YANG data template, except it has a representation with the media type "application/yang-patch+xml" or "application/yang-patch+json".

#### 1.1.6. Examples

Some protocol message lines within examples throughout the document are split into multiple lines for display purposes only. When a line ends with backslash ('\') as the last character, the line is wrapped

for display purposes. It is to be considered to be joined to the next line by deleting the backslash, the following line break, and the leading whitespace of the next line.

#### 1.1.7. Tree Diagram Notations

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

- o Brackets "[" and "]" enclose list keys.
- o Abbreviations before data node names: "rw" means configuration data (read-write), "ro" state data (read-only), and "x" operation resource (executable)
- o Symbols after data node names: "?" means an optional node 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. YANG Patch

A "YANG Patch" is an ordered list of edits that are applied to the target datastore by the RESTCONF server. The specific fields are defined in the YANG module in Section 3.

The YANG Patch operation is invoked by the RESTCONF client by sending a PATCH method request with a representation using either the "application/yang-patch+xml" or "application/yang-patch+json" media type. This message-body representing the YANG Patch input parameters MUST be present.

YANG Patch has some features that are not possible with the PATCH method in RESTCONF:

- o YANG Patch allows multiple sub-resources to be edited within the same PATCH method.
- o YANG Patch allows more precise edit operations than RESTCONF. There are 7 operations supported (create, delete, insert, merge, move, replace, remove).



- o YANG Patch uses an edit list with an explicit processing order. The edits are processed in client-specified order, and error processing can be precise even when multiple errors occur in the same patch request.

The YANG Patch "patch-id" may be useful for debugging, and SHOULD be present in any audit logging records generated by the RESTCONF server for a patch.

The RESTCONF server MUST return the Accept-Patch header field in an OPTIONS response, as specified in [RFC5789], which includes the media type for YANG Patch. This is needed by a client to determine the message encoding formats supported by the server (e.g., XML, JSON, or both). An example is shown in Figure 1.

Accept-Patch: application/yang-patch+xml,application/yang-patch+json

Figure 1: Example Accept-Patch header

Note that YANG Patch can only edit data resources. The PATCH method cannot be used to replace the datastore resource. Although the "ietf-yang-patch" YANG module is written using YANG version 1.1 [RFC7950], an implementation of YANG Patch can be used with content defined in YANG version 1 [RFC6020] as well.

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" [RFC7951]. If any meta-data needs to be sent in a JSON message, it is encoded according to "Defining and Using Metadata with YANG" [RFC7952].

## 2.1. Target Resource

The YANG Patch operation uses the RESTCONF target resource URI to identify the resource that will be patched. This can be the datastore resource itself, i.e., "{+restconf}/data", to edit top-level configuration data resources, or it can be a configuration data resource within the datastore resource, e.g., "{+restconf}/data/ietf-interfaces:interfaces", to edit sub-resources within a top-level configuration data resource.

The target resource MUST identify exactly one resource instance. If more than one resource instance is identified, then the request MUST NOT be processed, and a "400 Bad Request" error response MUST be sent by the server. If the target resource does not identify any existing resource instance then the request MUST NOT be processed, and a "404 Not Found" error response MUST be sent by the server.

Each edit with a YANG Patch identifies a target data node for the associated edit. This is described in Section 2.4.

## 2.2. yang-patch Request

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.

The merge, replace, create, delete, and remove edit operations have the exact same meaning as defined for the "operation" attribute in section 7.2 of [RFC6241].

Each edit within a YANG Patch MUST identify exactly one data resource instance. If an edit represents more than one resource instance, then the request MUST NOT be processed, and a "400 Bad Request" error response MUST be sent by the server. If the edit does not identify any existing resource instance, and the operation for the edit is not "create", then the request MUST NOT be processed, and a "404 Not Found" error response MUST be sent by the server. A "yang-patch-status" response MUST be sent by the server identifying the edit(s) that are not valid.

YANG Patch does not provide any access to specific datastores. It is an implementation detail how a server processes an edit if it is co-located with a NETCONF server that does provide access to individual datastores. A complete datastore cannot be replaced in the same manner as provided by the "copy-config" operation defined in section 7.3 of [RFC6241]. Only the specified nodes in a YANG Patch are affected.

A message-body representing the YANG Patch is sent by the RESTCONF 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

```
+----- yang-patch
+----- patch-id      string
+----- comment?     string
+----- edit* [edit-id]
+----- edit-id       string
+----- operation     enumeration
+----- target        target-resource-offset
+----- point?        target-resource-offset
+----- where?        enumeration
+----- value?
```

### 2.3. yang-patch-status Response

A message-body representing the YANG Patch Status is returned to the RESTCONF 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:

```

+----- yang-patch-status
+----- patch-id      string
+----- (global-status)?
|   +---:(global-errors)
|   |   +----- errors
|   |   |   +----- error*
|   |   |   |   +----- error-type      enumeration
|   |   |   |   +----- error-tag       string
|   |   |   |   +----- error-app-tag?  string
|   |   |   |   +----- error-path?    instance-identifier
|   |   |   |   +----- error-message?  string
|   |   |   |   +----- error-info?
|   |   +---:(ok)
|   |   |   +----- ok?                empty
+----- edit-status
+----- edit* [edit-id]
+----- edit-id      string
+----- (edit-status-choice)?
|   +---:(ok)
|   |   +----- ok?                empty
+---:(errors)
+----- errors
+----- error*
+----- error-type      enumeration
+----- error-tag       string
+----- error-app-tag?  string
+----- error-path?    instance-identifier
+----- error-message?  string
+----- error-info?

```

## 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 treated as an absolute path expression identifying the target data node for the corresponding edit. The first node specified in the "target" leaf is a top-level data node defined within a YANG module. The "target" leaf MUST NOT contain a single forward slash "/", since this would identify the datastore resource, not a data resource.

If the target resource specified in the request URI identifies a configuration data resource, then the path string in the "target" leaf is treated as a relative path expression. The first node specified in the "target" leaf is a child configuration data node of

the data node associated with the target resource. If the "target" leaf contains a single forward slash "/", then the target data node is the target resource data node.

## 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.

Operation	Description
create	create a new data resource if it does not already exist or error
delete	delete a data resource if it already exists or error
insert	insert a new user-ordered data resource
merge	merge the edit value with the target data resource; create if it does not already exist
move	re-order the target data resource
replace	replace the target data resource with the edit value
remove	remove a data resource if it already exists

YANG Patch Edit Operations

## 2.6. Successful Edit Response Handling

If a YANG Patch is completed without errors, the RESTCONF server MUST return a "yang-patch-status" message with a global-status choice set to 'ok'.

The RESTCONF server will save the running datastore to non-volatile storage if it supports non-volatile storage, and if the running datastore contents have changed, as specified in [I-D.ietf-netconf-restconf].

Refer to Appendix D.1.2 for a example of a successful YANG Patch response.

## 2.7. Error Handling

If a well-formed, schema-valid YANG Patch message is received, then the RESTCONF server will process the supplied edits in ascending order. The following error modes apply to the processing of this edit list:

If a YANG Patch is completed with errors, the RESTCONF server SHOULD return a "yang-patch-status" message. It is possible (e.g., within a distributed implementation), that an invalid request will be rejected before the YANG patch edits are processed. In this case, the server MUST send the appropriate HTTP error response instead.

Refer to Appendix D.1.1 for a example of an error YANG Patch response.

## 2.8. yang-patch RESTCONF Capability

A URI is defined to identify the YANG Patch extension to the base RESTCONF protocol. If the RESTCONF server supports the YANG Patch media type, then the "yang-patch" RESTCONF capability defined in Section 4.3 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 'yang-data' extension statements, which are not meant to be implemented as datastore contents by a RESTCONF server.

The "ietf-restconf" module from [I-D.ietf-netconf-restconf] is used by this module for the 'yang-data' extension definition.

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

<CODE BEGINS> file "ietf-yang-patch@2016-11-09.yang"

```
module ietf-yang-patch {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-patch";
  prefix "ypatch";

  import ietf-restconf { prefix rc; }

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

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

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

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

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

description

"This module contains conceptual YANG specifications  
for 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) 2016 IETF Trust and the persons identified as  
authors of the code. All rights reserved.

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

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

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

// RFC Ed.: remove this note  
// Note: extracted from draft-ietf-netconf-yang-patch-14.txt

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

```
revision 2016-11-09 {  
  description  
    "Initial revision.";  
  reference  
    "RFC XXXX: YANG Patch Media Type.";  
}
```

```
typedef target-resource-offset {  
  type string;  
  description  
    "Contains a data resource identifier string representing  
    a sub-resource within the target resource.
```

The document root for this expression is the target resource that is specified in the protocol operation (e.g., the URI for the PATCH request).

This string is encoded according the same rules as a data resource identifier in a RESTCONF Request URI.";

```
// RFC Ed.: replace "draft-ietf-netconf-restconf" below
// with RFC XXXX, where XXXX is the number of the RESTCONF RFC,
// and remove this note.
```

```
reference
  "draft-ietf-netconf-restconf, section 3.5.3";
}

rc:yang-data "yang-patch" {
  uses yang-patch;
}

rc:yang-data "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 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.";

    reference
```



```
"RFC 7950, section 8.3.";

leaf patch-id {
  type string;
  mandatory true;
  description
    "An arbitrary string provided by the client to identify
    the entire patch.  Error messages returned by the server
    pertaining to this patch will be identified by this
    patch-id value.  A client SHOULD attempt to generate
    unique patch-id values to distinguish between transactions
    from multiple clients in any audit logs maintained
    by the server.";
}

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. The
                'target' leaf identifies the data node to be created,
                not the parent data node.";
            }
            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 node for the edit
        operation. If the target has the value '/', then
        the target data node is the target resource.
        The target node MUST identify a data resource,
        not the datastore resource.";
}

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.";
}

anydata 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 anydata 'value'
        contains the target resource associated with the
        'target' leaf.

        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 contains one instance of foo:

        <value>
            <foo xmlns='example-foo-namespace'>

```

```

        <a>some value</a>
        <b>42</b>
    </foo>
</value>
";
    }
}
}

} // grouping yang-patch

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.
                If there was no patch-id present in the request
                then this field will not be present.";
        }

        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 that are 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";

    list edit {
        key edit-id;

        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.";

        leaf edit-id {
            type string;
            description
                "Response status is for the edit list entry
                with this edit-id value.";
        }
    }
    choice edit-status-choice {
        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  
  
}
```

<CODE ENDS>

## 4. IANA Considerations

### 4.1. YANG Module Registry

This document registers one URI as a namespace in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made.

URI: urn:ietf:params:xml:ns:yang:ietf-yang-patch  
Registrant Contact: The NETCONF 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  
namespace: urn:ietf:params:xml:ns:yang:ietf-yang-patch  
prefix: ypatch  
// RFC Ed.: replace XXXX with RFC number and remove this note  
reference: RFC XXXX

### 4.2. Media Types

#### 4.2.1. Media Type application/yang-patch+xml

Type name: application  
Subtype name: yang-patch+xml  
Required parameters: None  
Optional parameters: None  
  
// RFC Ed.: replace 'XXXX' with the real RFC number,  
// and remove this note  
  
Encoding considerations: 8-bit

The utf-8 charset is always used for this type.  
Each conceptual YANG data node is encoded according to the XML Encoding Rules and Canonical Format for the specific YANG data node type defined in [RFC7950].  
In addition, the "yang-patch" YANG Patch template found in [RFCXXXX] defines the structure of a YANG Patch request.

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

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

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

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

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

Published specification: RFC XXXX

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

Fragment identifier considerations: Same as for application/xml

Additional information:

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

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

Person & email address to contact for further information: See



Authors' Addresses section of [RFCXXXX].

Intended usage: COMMON

Restrictions on usage: N/A

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

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

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

Provisional registration? (standards tree only): no

#### 4.2.2. Media Type application/yang-patch+json

Type name: application

Subtype name: yang-patch+json

Required parameters: None

Optional parameters: None

// RFC Ed.: replace draft-ietf-netmod-yang-json with  
// the actual RFC reference for JSON Encoding of YANG Data,  
// and remove this note.

// RFC Ed.: replace draft-ietf-netmod-yang-metadata with  
// the actual RFC reference for JSON Encoding of YANG Data,  
// and remove this note.

// RFC Ed.: replace 'XXXX' with the real RFC number,  
// and remove this note

Encoding considerations: 8-bit

The utf-8 charset is always used for this type.  
Each conceptual YANG data node is encoded according to  
[draft-ietf-netmod-yang-json]. A data annotation is  
encoded according to [draft-ietf-netmod-yang-metadata]  
In addition, the "yang-patch" YANG Patch template found  
in [RFCXXXX] defines the structure of a YANG Patch request.

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

```
// RFC number, and remove this note.

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

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

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

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

Published specification: RFC XXXX

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

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

Additional information:

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

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

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

Intended usage: COMMON

Restrictions on usage: N/A

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

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

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

Provisional registration? (standards tree only): no

#### 4.3. 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. Message integrity is provided by the RESTCONF protocol. There is no additional capability to validate that a patch has not been altered.

It may be possible to use YANG Patch with other protocols besides RESTCONF, which is outside the scope of this document.

For RESTCONF, both the client and server MUST be authenticated, according to section 2 of [I-D.ietf-netconf-restconf]. It is important for RESTCONF 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 PATCH request MUST be applied atomically, as specified in section 2 of [RFC5789].

A RESTCONF server implementation SHOULD attempt to prevent system disruption due to incremental processing of the YANG Patch edit list. It may be possible to construct an attack on such a RESTCONF server, which relies on the edit processing order mandated by YANG Patch. A server SHOULD apply only the fully validated configuration to the underlying system. For example, an edit list which deleted an interface and then recreated it could cause system disruption if the edit list was incrementally applied.

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

## 6. Normative References

- [I-D.ietf-netconf-restconf]  
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-18 (work in progress), October 2016.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, March 2010.
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<http://www.rfc-editor.org/info/rfc7159>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<http://www.rfc-editor.org/info/rfc7230>>.
- [RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.

- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<http://www.rfc-editor.org/info/rfc7951>>.
- [RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <<http://www.rfc-editor.org/info/rfc7952>>.
- [W3C.REC-xml-20081126]  
Yergeau, F., Maler, E., Paoli, J., Sperberg-McQueen, C., and T. Bray, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <<http://www.w3.org/TR/2008/REC-xml-20081126>>.

#### 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. v12 to v13

- o clarifications based on IESG reviews

##### B.2. v11 to v12

- o clarify target resource must exist
- o fix errors in some examples
- o change application/yang-patch-xml to application/yang-patch+xml
- o clarified some section titles

- o clarified error responses for multiple edit instances
- o made patch-id field mandatory
- o referenced NETCONF operation attribute

#### B.3. v10 to v11

- o change application/yang-patch to application/yang-patch-xml
- o change server to RESTCONF server and remove NETCONF server term
- o change client to RESTCONF client and remove NETCONF client term
- o clarified that YANG 1.0 content can be used in a YANG Patch implementation
- o clarified more terminology
- o fixed missing keys in edit examples
- o added insert list example

#### B.4. v09 to v10

- o change yang-patch+xml to yang-patch
- o clarify application/yang-patch+json media type
- o add edit datastore example
- o change data-resource-offset typedef so it is consistent for XML and JSON

#### B.5. v08 to v09

- o change RFC 7158 reference to RFC 7159 reference
- o change RFC 2616 reference to RFC 7230 reference
- o remove unused HTTP terms
- o remove import-by-revision of ietf-restconf; not needed
- o change application/yang.patch media type to application/yang-patch
- o remove application/yang.patch-status media type; use application/yang-data instead

## B.6. v07 to v08

- o clarified target datastore and target data node terms
- o clarified that target leaf can be single forward slash '/'
- o added Successful edit response handling section
- o clarified that YANG Patch draft is for RESTCONF protocol only but may be defined for other protocols outside this document
- o clarified that YANG Patch draft is for configuration datastores only but may be defined for other datastore types outside this document
- o fixed typos

## B.7. v06 to v07

- o converted YANG module to YANG 1.1
- o changed anyxml value to anydata value
- o updated import revision date for ietf-restconf
- o updated revision date for ietf-yang-patch because import-by-revision date needed to be changed

## B.8. v05 to v06

- o changed errors example so a full request and error response is shown in XML format
- o fixed error-path to match instance-identifier encoding for both XML and JSON
- o added references for YANG to JSON and YANG Metadata drafts
- o clarified that YANG JSON drafts are used for encoding, not plain JSON

## B.9. v04 to v05

- o updated reference to RESTCONF

## B.10. v03 to v04

- o removed NETCONF specific text
- o changed data-resource-offset typedef from a relative URI to an XPath absolute path expression
- o clarified insert operation
- o removed requirement that edits MUST be applied in ascending order
- o change SHOULD keep datastore unchanged on error to MUST (this is required by HTTP PATCH)
- o removed length restriction on 'comment' leaf
- o updated YANG tree for example-jukebox library

## B.11. v02 to v03

- o added usage of restconf-media-type extension to map the yang-patch and yang-patch-status groupings to media types
- o added yang-patch RESTCONF capability URI
- o Added sub-section for terms used from RESTCONF
- o filled in security considerations section

## B.12. v01 to v02

- o Reversed order of change log
- o Clarified anyxml structure of "value" parameter within a YANG patch request (github issue #1)
- o Updated RESTCONF reference
- o Added note to open issues section to check github instead

## B.13. v00 to v01

- o Added text requiring support for Accept-Patch header field, and removed 'Identification of YANG Patch capabilities' open issue.
- o Removed 'location' leaf from yang-patch-status grouping



- o Removed open issue 'Protocol independence' because the location leaf was removed.
- o 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.14. birman: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]
    |   |   |   +--rw name      string
    |   |   |   +--rw genre?   identityref
    |   |   |   +--rw year?    uint16
    |   |   |   +--rw admin
    |   |   |   |   +--rw label?          string
    |   |   |   |   +--rw catalogue-number? string
    |   |   |   +--rw song* [name]
    |   |   |   |   +--rw name      string
    |   |   |   |   +--rw location  string
    |   |   |   |   +--rw format?   string
    |   |   |   |   +--rw length?   uint32
    |   |   +--ro artist-count? uint32
    |   |   +--ro album-count?  uint32
    |   |   +--ro song-count?   uint32
    |   +--rw playlist* [name]
    |   |   +--rw name      string
    |   |   +--rw description? string
    |   |   +--rw song* [index]
    |   |   |   +--rw index  uint32
    |   |   |   +--rw id     leafref
    |   +--rw player
    |   |   +--rw gap?   decimal64

```

rpcs:

```

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

```

## D.1. YANG Patch Examples

This section includes RESTCONF examples. Most examples are shown in JSON encoding [RFC7159], 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 the RESTCONF client:

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

<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=Bridge%20Burning</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=Rope</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=Dear%20Rosemary</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 the RESTCONF 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-data+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>
</yang-patch-status>
```

JSON Response from the RESTCONF 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.

```

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-data+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"
              }
            ]
          }
        }
      ]
    }
  }
}

```

#### D.1.2. Add Resources: Success

The following example shows several songs being added to an existing album.

- o Each of 2 edits contains one song.
- o Both edits succeed and new sub-resources are created

Request from the RESTCONF client:

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

```
{
  "ietf-yang-patch:yang-patch" : {
    "patch-id" : "add-songs-patch-2",
    "edit" : [
      {
        "edit-id" : "edit1",
        "operation" : "create",
        "target" : "/song=Rope",
        "value" : {
          "song" : [
            {
              "name" : "Rope",
              "location" : "/media/rope.mp3",
              "format" : "MP3",
              "length" : 259
            }
          ]
        }
      },
      {
        "edit-id" : "edit2",
        "operation" : "create",
        "target" : "/song=Dear%20Rosemary",
        "value" : {
          "song" : [
            {
              "name" : "Dear Rosemary",
              "location" : "/media/dear_rosemary.mp3",
              "format" : "MP3",
              "length" : 269
            }
          ]
        }
      }
    ]
  }
}
```

Response from the RESTCONF server:

```
HTTP/1.1 200 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-data+json
```

```
{
  "ietf-yang-patch:yang-patch-status" : {
    "patch-id" : "add-songs-patch-2",
    "ok" : [null]
  }
}
```

#### D.1.3. Insert list entry example

The following example shows a song being inserted within an existing playlist. Song "6" in playlist "Foo-One" is being inserted after song "5" in the playlist. The operation succeeds, so a non-error reply example can be shown.



Request from the RESTCONF client:

```
PATCH /restconf/data/example-jukebox:jukebox/\
  playlist=Foo-One HTTP/1.1
Host: example.com
Accept: application/yang-data+json
Content-Type: application/yang-patch+json

{
  "ietf-yang-patch:yang-patch" : {
    "patch-id" : "move-song-patch",
    "comment" : "Insert song 6 after song 5",
    "edit" : [
      {
        "edit-id" : "edit1",
        "operation" : "insert",
        "target" : "/song=6",
        "point" : "/song=5",
        "where" : "after",
        "value" : {
          "example-jukebox:song" : [
            {
              "name" : "Dear Prudence",
              "location" : "/media/dear_prudence.mp3",
              "format" : "MP3",
              "length" : 236
            }
          ]
        }
      }
    ]
  }
}
```

Response from the RESTCONF server:

```
HTTP/1.1 200 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-data+json

{
  "ietf-yang-patch:yang-patch-status" : {
    "patch-id" : "move-song-patch",
    "ok" : [null]
  }
}
```

## D.1.4. 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. Note that no "value" parameter is needed for a "move" operation. The operation succeeds, so a non-error reply example can be shown.

Request from the RESTCONF client:

```
PATCH /restconf/data/example-jukebox:jukebox/\
  playlist=Foo-One HTTP/1.1
Host: example.com
Accept: application/yang-data+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" : "/song=3",
        "where" : "after"
      }
    ]
  }
}
```

Response from the RESTCONF server:

```
HTTP/1.1 200 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-data+json

{
  "ietf-restconf:yang-patch-status" : {
    "patch-id" : "move-song-patch",
    "ok" : [null]
  }
}
```

#### D.1.5. Edit datastore resource example

The following example shows how 3 top-level data nodes from different modules can be edited at the same time.

Example module "foo" defines leaf X. Example module "bar" defines container Y, with child leafs A and B. Example module "baz" defines list Z, with key C and child leafs D and E.

Request from the RESTCONF client:

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

{
  "ietf-yang-patch:yang-patch" : {
    "patch-id" : "datastore-patch-1",
    "comment" : "Edit 3 top-level data nodes at once",
    "edit" : [
      {
        "edit-id" : "edit1",
        "operation" : "create",
        "target" : "/foo:X",
        "value" : {
          "foo:X" : 42
        }
      },
      {
        "edit-id" : "edit2",
        "operation" : "merge",
        "target" : "/bar:Y",
        "value" : {
          "bar:Y" : {
            "A" : "test1",
            "B" : 99
          }
        }
      },
      {
        "edit-id" : "edit3",
        "operation" : "replace",
        "target" : "/baz:Z=2",
        "value" : {
          "baz:Z" : [
            {
              "C" : 2,
              "D" : 100,
              "E" : false
            }
          ]
        }
      }
    ]
  }
}
```

Response from the RESTCONF server:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 13:02:20 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
Content-Type: application/yang-data+json
```

```
{
  "ietf-yang-patch:yang-patch-status" : {
    "patch-id" : "datastore-patch-1",
    "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

NETCONF Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: July 19, 2019

K. Watsen  
Juniper Networks  
M. Abrahamsson  
T-Systems  
I. Farrer  
Deutsche Telekom AG  
January 15, 2019

Secure Zero Touch Provisioning (SZTP)  
draft-ietf-netconf-zerotouch-29

Abstract

This draft presents a technique to securely provision a networking device when it is booting in a factory-default state. Variations in the solution enables it to be used on both public and private networks. The provisioning steps are able to update the boot image, commit an initial configuration, and execute arbitrary scripts to address auxiliary needs. The updated device is subsequently able to establish secure connections with other systems. For instance, a device may establish NETCONF (RFC 6241) and/or RESTCONF (RFC 8040) connections with deployment-specific network management systems.

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. No other RFC Editor instructions are specified elsewhere in this document.

Artwork in the IANA Considerations section contains placeholder values for DHCP options pending IANA assignment. Please apply the following replacements:

- o "TBD1" --> the assigned value for id-ct-sztpConveyedInfoXML
- o "TBD2" --> the assigned value for id-ct-sztpConveyedInfoJSON
- o "TBD\_IANA\_URL" --> the assigned URL for the IANA registry

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

- o "XXXX" --> the assigned numerical RFC value for this draft

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

- o "2019-01-15" --> the publication date of this draft

The following one Appendix section is to be removed prior to publication:

- o Appendix D. Change Log

#### 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 <https://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 July 19, 2019.

#### Copyright Notice

Copyright (c) 2019 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 (<https://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. Use Cases . . . . .	5
1.2. Terminology . . . . .	6
1.3. Requirements Language . . . . .	8
1.4. Tree Diagrams . . . . .	8
2. Types of Conveyed Information . . . . .	8
2.1. Redirect Information . . . . .	8

2.2. Onboarding Information . . . . .	9
3. Artifacts . . . . .	10
3.1. Conveyed Information . . . . .	10
3.2. Owner Certificate . . . . .	11
3.3. Ownership Voucher . . . . .	12
3.4. Artifact Encryption . . . . .	13
3.5. Artifact Groupings . . . . .	13
4. Sources of Bootstrapping Data . . . . .	14
4.1. Removable Storage . . . . .	15
4.2. DNS Server . . . . .	16
4.3. DHCP Server . . . . .	19
4.4. Bootstrap Server . . . . .	20
5. Device Details . . . . .	21
5.1. Initial State . . . . .	21
5.2. Boot Sequence . . . . .	23
5.3. Processing a Source of Bootstrapping Data . . . . .	25
5.4. Validating Signed Data . . . . .	26
5.5. Processing Redirect Information . . . . .	27
5.6. Processing Onboarding Information . . . . .	28
6. The Conveyed Information Data Model . . . . .	31
6.1. Data Model Overview . . . . .	31
6.2. Example Usage . . . . .	32
6.3. YANG Module . . . . .	34
7. The SZTP Bootstrap Server API . . . . .	40
7.1. API Overview . . . . .	40
7.2. Example Usage . . . . .	41
7.3. YANG Module . . . . .	44
8. DHCP Options . . . . .	56
8.1. DHCPv4 SZTP Redirect Option . . . . .	56
8.2. DHCPv6 SZTP Redirect Option . . . . .	57
8.3. Common Field Encoding . . . . .	58
9. Security Considerations . . . . .	59
9.1. Clock Sensitivity . . . . .	59
9.2. Use of IDevID Certificates . . . . .	59
9.3. Immutable Storage for Trust Anchors . . . . .	59
9.4. Secure Storage for Long-lived Private Keys . . . . .	59
9.5. Blindly Authenticating a Bootstrap Server . . . . .	60
9.6. Disclosing Information to Untrusted Servers . . . . .	60
9.7. Sequencing Sources of Bootstrapping Data . . . . .	61
9.8. Safety of Private Keys used for Trust . . . . .	61
9.9. Increased Reliance on Manufacturers . . . . .	62
9.10. Concerns with Trusted Bootstrap Servers . . . . .	62
9.11. Validity Period for Conveyed Information . . . . .	63
9.12. Cascading Trust via Redirects . . . . .	64
9.13. Possible Reuse of Private Keys . . . . .	64
9.14. Non-Issue with Encrypting Signed Artifacts . . . . .	65
9.15. The "ietf-sztp-conveyed-info" YANG Module . . . . .	65
9.16. The "ietf-sztp-bootstrap-server" YANG Module . . . . .	66



10. IANA Considerations . . . . .	66
10.1. The IETF XML Registry . . . . .	66
10.2. The YANG Module Names Registry . . . . .	67
10.3. The SMI Security for S/MIME CMS Content Type Registry . . . . .	67
10.4. The BOOTP Manufacturer Extensions and DHCP Options Registry . . . . .	67
10.5. The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Registry . . . . .	68
10.6. The Service Name and Transport Protocol Port Number Registry . . . . .	68
10.7. The DNS Underscore Global Scoped Entry Registry . . . . .	68
11. References . . . . .	69
11.1. Normative References . . . . .	69
11.2. Informative References . . . . .	71
Appendix A. Example Device Data Model . . . . .	74
A.1. Data Model Overview . . . . .	74
A.2. Example Usage . . . . .	74
A.3. YANG Module . . . . .	75
Appendix B. Promoting a Connection from Untrusted to Trusted . . . . .	78
Appendix C. Workflow Overview . . . . .	80
C.1. Enrollment and Ordering Devices . . . . .	80
C.2. Owner Stages the Network for Bootstrap . . . . .	82
C.3. Device Powers On . . . . .	84
Appendix D. Change Log . . . . .	87
D.1. ID to 00 . . . . .	87
D.2. 00 to 01 . . . . .	87
D.3. 01 to 02 . . . . .	87
D.4. 02 to 03 . . . . .	88
D.5. 03 to 04 . . . . .	88
D.6. 04 to 05 . . . . .	88
D.7. 05 to 06 . . . . .	89
D.8. 06 to 07 . . . . .	89
D.9. 07 to 08 . . . . .	89
D.10. 08 to 09 . . . . .	89
D.11. 09 to 10 . . . . .	89
D.12. 10 to 11 . . . . .	90
D.13. 11 to 12 . . . . .	90
D.14. 12 to 13 . . . . .	90
D.15. 13 to 14 . . . . .	91
D.16. 14 to 15 . . . . .	91
D.17. 15 to 16 . . . . .	91
D.18. 16 to 17 . . . . .	92
D.19. 17 to 18 . . . . .	92
D.20. 18 to 19 . . . . .	93
D.21. 19 to 20 . . . . .	93
D.22. 20 to 21 . . . . .	94
D.23. 21 to 22 . . . . .	94
D.24. 22 to 23 . . . . .	94

D.25. 23 to 24	95
D.26. 24 to 25	95
D.27. 25 to 26	96
D.28. 26 to 27	96
D.29. 27 to 28	97
Acknowledgements	97
Authors' Addresses	97

## 1. Introduction

A fundamental business requirement for any network operator 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 for installations is both cost prohibitive and does not scale.

This document defines Secure Zero Touch Provisioning (SZTP), a bootstrapping strategy enabling devices to securely obtain bootstrapping data with no installer action beyond physical placement and connecting network and power cables. As such, SZTP enables non-technical personnel to bring up devices in remote locations without the need for any operator input.

The SZTP solution includes updating the boot image, committing an initial configuration, and executing arbitrary scripts to address auxiliary needs. The updated device is subsequently able to establish secure connections with other systems. For instance, a device may establish NETCONF [RFC8040] and/or RESTCONF [RFC6241] connections with deployment-specific network management systems.

This document primarily regards physical devices, where the setting of the device's initial state, described in Section 5.1, occurs during the device's manufacturing process. The SZTP solution may be extended to support virtual machines or other such logical constructs, but details for how this can be accomplished is left for future work.

### 1.1. Use Cases

#### o Device 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 to provide any bootstrapping support, and with no other nearby device to leverage, the device has no recourse but to reach out to an Internet-based bootstrap server to bootstrap from.

- o Device 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 from. If no such information is available, or the device is unable to use the information provided, it can then reach out to the network just as it would for the remotely administered network use-case.

Conceptual workflows for how SZTP might be deployed are provided in Appendix C.

## 1.2. Terminology

This document uses the following terms (sorted by name):

**Artifact:** The term "artifact" is used throughout to represent any of the three artifacts defined in Section 3 (conveyed information, ownership voucher, and owner certificate). These artifacts collectively provide all the bootstrapping data a device may use.

**Bootstrapping Data:** The term "bootstrapping data" is used throughout this document to refer to the collection of data that a device may obtain during the bootstrapping process. Specifically, it refers to the three artifacts conveyed information, owner certificate, and ownership voucher, as described in Section 3.

**Bootstrap Server:** The term "bootstrap server" is used within this document to mean any RESTCONF server implementing the YANG module defined in Section 7.3.

**Conveyed Information:** The term "conveyed information" is used herein to refer either redirect information or onboarding information. Conveyed information is one of the three bootstrapping artifacts described in Section 3.

**Device:** The term "device" is used throughout this document to refer to a network element that needs to be bootstrapped. See Section 5 for more information about devices.

**Manufacturer:** The term "manufacturer" is used herein to refer to the manufacturer of a device or a delegate of the manufacturer.

**Network Management System (NMS):** The acronym "NMS" is used throughout this document to refer to the deployment-specific management system that the bootstrapping process is responsible for introducing devices to. From a device's perspective, when

the bootstrapping process has completed, the NMS is a NETCONF or RESTCONF client.

**Onboarding Information:** The term "onboarding information" is used herein to refer to one of the two types of "conveyed information" defined in this document, the other being "redirect information". Onboarding information is formally defined by the "onboarding-information" YANG-data structure in Section 6.3.

**Onboarding Server:** The term "onboarding server" is used herein to refer to a bootstrap server that only returns onboarding information.

**Owner:** The term "owner" is used throughout this document to refer to the person or organization that purchased or otherwise owns a device.

**Owner Certificate:** The term "owner certificate" is used in this document to represent an X.509 certificate that binds an owner identity to a public key, which a device can use to validate a signature over the conveyed information artifact. The owner certificate may be communicated along with its chain of intermediate certificates leading up to a known trust anchor. The owner certificate is one of the three bootstrapping artifacts described in Section 3.

**Ownership Voucher:** The term "ownership voucher" is used in this document to represent the voucher artifact defined in [RFC8366]. The ownership voucher is used to assign a device to an owner. The ownership voucher is one of the three bootstrapping artifacts described in Section 3.

**Redirect Information:** The term "redirect information" is used herein to refer to one of the two types of "conveyed information" defined in this document, the other being "onboarding information". Redirect information is formally defined by the "redirect-information" YANG-data structure in Section 6.3.

**Redirect Server:** The term "redirect server" is used to refer to a bootstrap server that only returns redirect information. A redirect server is particularly useful when hosted by a manufacturer, as a well-known (e.g., Internet-based) resource to redirect devices to deployment-specific bootstrap servers.

**Signed Data:** The term "signed data" is used throughout to mean conveyed information that has been signed, specifically by a private key possessed by a device's owner.

**Unsigned Data:** The term "unsigned data" is used throughout to mean conveyed information that has not been signed.

### 1.3. Requirements Language

The key words "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] [RFC8174] when, and only when, they appear in all capitals, as shown here.

### 1.4. Tree Diagrams

Tree diagrams used in this document follow the notation defined in [RFC8340].

## 2. Types of Conveyed Information

This document defines two types of conveyed information that devices can access during the bootstrapping process. These conveyed information types are described in this section. Examples are provided in Section 6.2

### 2.1. Redirect Information

Redirect information redirects a device to another bootstrap server. Redirect information encodes a list of bootstrap servers, each specifying the bootstrap server's hostname (or IP address), an optional port, and an optional trust anchor certificate that the device can use to authenticate the bootstrap server with.

Redirect information is YANG modeled data formally defined by the "redirect-information" container in the YANG module presented in Section 6.3. This container has the tree diagram shown below.

```
+--:(redirect-information)
  +-- redirect-information
    +-- bootstrap-server* [address]
      +-- address          inet:host
      +-- port?           inet:port-number
      +-- trust-anchor?   cms
```

Redirect information may be trusted or untrusted. The redirect information is trusted whenever it is obtained via a secure connection to a trusted bootstrap server, or whenever it is signed by the device's owner. In all other cases, the redirect information is untrusted.

Trusted redirect information is useful for enabling a device to establish a secure connection to a specified bootstrap server, which is possible when the redirect information includes the bootstrap server's trust anchor certificate.

Untrusted redirect information is useful for directing a device to a bootstrap server where signed data has been staged for it to obtain. Note that, when the redirect information is untrusted, devices discard any potentially included trust anchor certificates.

How devices process redirect information is described in Section 5.5.

## 2.2. Onboarding Information

Onboarding information provides data necessary for a device to bootstrap itself and establish secure connections with other systems. As defined in this document, onboarding information can specify details about the boot image a device must be running, specify an initial configuration the device must commit, and specify scripts that the device must successfully execute.

Onboarding information is YANG modeled data formally defined by the "onboarding-information" container in the YANG module presented in Section 6.3. This container has the tree diagram shown below.

```
+--:(onboarding-information)
  +-- onboarding-information
    +-- boot-image
      +-- os-name?          string
      +-- os-version?      string
      +-- download-uri*    inet:uri
      +-- image-verification* [hash-algorithm]
        +-- hash-algorithm identityref
        +-- hash-value     yang:hex-string
    +-- configuration-handling? enumeration
    +-- pre-configuration-script? script
    +-- configuration?      binary
    +-- post-configuration-script? script
```

Onboarding information must be trusted for it to be of any use to a device. There is no option for a device to process untrusted onboarding information.

Onboarding information is trusted whenever it is obtained via a secure connection to a trusted bootstrap server, or whenever it is signed by the device's owner. In all other cases, the onboarding information is untrusted.

How devices process onboarding information is described in Section 5.6.

### 3. Artifacts

This document defines three artifacts that can be made available to devices while they are bootstrapping. Each source of bootstrapping data specifies how it provides the artifacts defined in this section (see Section 4).

#### 3.1. Conveyed Information

The conveyed information artifact encodes the essential bootstrapping data for the device. This artifact is used to encode the redirect information and onboarding information types discussed in Section 2.

The conveyed information artifact is a CMS structure, as described in [RFC5652], encoded using ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690 [ITU.X690.2015]. The CMS structure MUST contain content conforming to the YANG module specified in Section 6.3.

The conveyed information CMS structure may encode signed or unsigned bootstrapping data. When the bootstrapping data is signed, it may also be encrypted but, from a terminology perspective, it is still "signed data" Section 1.2.

When the conveyed information artifact is unsigned, as it might be when communicated over trusted channels, the CMS structure's top-most content type MUST be one of the OIDs described in Section 10.3 (i.e., id-ct-sztpConveyedInfoXML or id-ct-sztpConveyedInfoJSON), or the OID id-data (1.2.840.113549.1.7.1). When the OID id-data is used, the encoding (JSON, XML, etc.) SHOULD be communicated externally. In either case, the associated content is an octet string containing "conveyed-information" data in the expected encoding.

When the conveyed information artifact is unsigned and encrypted, as it might be when communicated over trusted channels but, for some reason, the operator wants to ensure that only the device is able to see the contents, the CMS structure's top-most content type MUST be the OID id-envelopedData (1.2.840.113549.1.7.3). Furthermore, the encryptedContentInfo's content type MUST be one of the OIDs described in Section 10.3 (i.e., id-ct-sztpConveyedInfoXML or id-ct-sztpConveyedInfoJSON), or the OID id-data (1.2.840.113549.1.7.1). When the OID id-data is used, the encoding (JSON, XML, etc.) SHOULD be communicated externally. In either case, the associated content is an octet string containing "conveyed-information" data in the expected encoding.

When the conveyed information artifact is signed, as it might be when communicated over untrusted channels, the CMS structure's top-most content type MUST be the OID `id-signedData` (1.2.840.113549.1.7.2). Furthermore, the inner `eContentType` MUST be one of the OIDs described in Section 10.3 (i.e., `id-ct-sztpConveyedInfoXML` or `id-ct-sztpConveyedInfoJSON`), or the OID `id-data` (1.2.840.113549.1.7.1). When the OID `id-data` is used, the encoding (JSON, XML, etc.) SHOULD be communicated externally. In either case, the associated content or `eContent` is an octet string containing "conveyed-information" data in the expected encoding.

When the conveyed information artifact is signed and encrypted, as it might be when communicated over untrusted channels and privacy is important, the CMS structure's top-most content type MUST be the OID `id-envelopedData` (1.2.840.113549.1.7.3). Furthermore, the `encryptedContentInfo`'s content type MUST be the OID `id-signedData` (1.2.840.113549.1.7.2), whose `eContentType` MUST be one of the OIDs described in Section 10.3 (i.e., `id-ct-sztpConveyedInfoXML` or `id-ct-sztpConveyedInfoJSON`), or the OID `id-data` (1.2.840.113549.1.7.1). When the OID `id-data` is used, the encoding (JSON, XML, etc.) SHOULD be communicated externally. In either case, the associated content or `eContent` is an octet string containing "conveyed-information" data in the expected encoding.

### 3.2. Owner Certificate

The owner certificate artifact is an X.509 certificate [RFC5280] that is used to identify an "owner" (e.g., an organization). The owner certificate can be signed by any certificate authority (CA). The owner certificate either MUST have no Key Usage specified or the Key Usage MUST at least set the "digitalSignature" bit. The values for the owner certificate's "subject" and/or "subjectAltName" are not constrained by this document.

The owner certificate is used by a device to verify the signature over the conveyed information artifact (Section 3.1) that the device should have also received, as described in Section 3.5. In particular, the device verifies the signature using the public key in the owner certificate over the content contained within the conveyed information artifact.

The owner certificate artifact is formally a CMS structure, as specified by [RFC5652], encoded using ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690 [ITU.X690.2015].

The owner certificate CMS structure MUST contain the owner certificate itself, as well as all intermediate certificates leading to the "pinned-domain-cert" certificate specified in the ownership



voucher. The owner certificate artifact MAY optionally include the "pinned-domain-cert" as well.

In order to support devices deployed on private networks, the owner certificate CMS structure MAY also contain suitably fresh, as determined by local policy, revocation objects (e.g., CRLs). Having these revocation objects stapled to the owner certificate may obviate the need for the device to have to download them dynamically using the CRL distribution point or an OCSP responder specified in the associated certificates.

When unencrypted, the owner certificate artifact's CMS structure's top-most content type MUST be the OID id-signedData (1.2.840.113549.1.7.2). The inner SignedData structure is the degenerate form, whereby there are no signers, that is commonly used to disseminate certificates and revocation objects.

When encrypted, the owner certificate artifact's CMS structure's top-most content type MUST be the OID id-envelopedData (1.2.840.113549.1.7.3), and the encryptedContentInfo's content type MUST be the OID id-signedData (1.2.840.113549.1.7.2), whereby the inner SignedData structure is the degenerate form that has no signers commonly used to disseminate certificates and revocation objects.

### 3.3. Ownership Voucher

The ownership voucher artifact is used to securely identify a device's owner, as it is known to the manufacturer. The ownership voucher is signed by the device's manufacturer.

The ownership voucher is used to verify the owner certificate (Section 3.2) that the device should have also received, as described in Section 3.5. In particular, the device verifies that the owner certificate has a chain of trust leading to the trusted certificate included in the ownership voucher ("pinned-domain-cert"). Note that this relationship holds even when the owner certificate is a self-signed certificate, and hence also the pinned-domain-cert.

When unencrypted, the ownership voucher artifact is as defined in [RFC8366]. As described, it is a CMS structure whose top-most content type MUST be the OID id-signedData (1.2.840.113549.1.7.2), whose eContentType MUST be OID id-ct-animaJSONVoucher (1.2.840.113549.1.9.16.1), or the OID id-data (1.2.840.113549.1.7.1). When the OID id-data is used, the encoding (JSON, XML, etc.) SHOULD be communicated externally. In either case, the associated content is an octet string containing ietf-voucher data in the expected encoding.

When encrypted, the ownership voucher artifact's CMS structure's top-most content type MUST be the OID id-envelopedData (1.2.840.113549.1.7.3), and the encryptedContentInfo's content type MUST be the OID id-signedData (1.2.840.113549.1.7.2), whose eContentType MUST be OID id-ct-animaJSONVoucher (1.2.840.113549.1.9.16.1), or the OID id-data (1.2.840.113549.1.7.1). When the OID id-data is used, the encoding (JSON, XML, etc.) SHOULD be communicated externally. In either case, the associated content is an octet string containing ietf-voucher data in the expected encoding.

### 3.4. Artifact Encryption

Each of the three artifacts MAY be individually encrypted. Encryption may be important in some environments where the content is considered sensitive.

Each of the three artifacts are encrypted in the same way, by the unencrypted form being encapsulated inside a CMS EnvelopedData type.

As a consequence, both the conveyed information and ownership voucher artifacts are signed and then encrypted, never encrypted and then signed.

This sequencing has the advantage of shrouding the signer's certificate, and ensuring that the owner knows the content being signed. This sequencing further enables the owner to inspect an unencrypted voucher obtained from a manufacturer and then encrypt the voucher later themselves, perhaps while also stapling in current revocation objects, when ready to place the artifact in an unsafe location.

When encrypted, the CMS MUST be encrypted using a secure device identity certificate for the device. This certificate MAY be the same as the TLS-level client certificate the device uses when connecting to bootstrap servers. The owner must possess the device's identity certificate at the time of encrypting the data. How the owner comes to possess the device's identity certificate for this purpose is outside the scope of this document.

### 3.5. Artifact Groupings

The previous sections discussed the bootstrapping artifacts, but only certain groupings of these artifacts make sense to return in the various bootstrapping situations described in this document. These groupings are:

Unsigned Data: This artifact grouping is useful for cases when transport level security can be used to convey trust (e.g., HTTPS), or when the conveyed information can be processed in a provisional manner (i.e. unsigned redirect information).

Signed Data, without revocations: This artifact grouping is useful when signed data is needed (i.e., because the data is obtained from an untrusted source and it cannot be processed provisionally) and either revocations are not needed or the revocations can be obtained dynamically.

Signed Data, with revocations: This artifact grouping is useful when signed data is needed (i.e., because the data is obtained from an untrusted source and it cannot be processed provisionally), and revocations are needed, and the revocations cannot be obtained dynamically.

The presence of each artifact, and any distinguishing characteristics, are identified for each artifact grouping in the table below ("yes/no" regards if the artifact is present in the artifact grouping):

Artifact Grouping	Conveyed Information	Ownership Voucher	Owner Certificate
Unsigned Data	Yes, no sig	No	No
Signed Data, without revocations	Yes, with sig	Yes, without revocations	Yes, without revocations
Signed Data, with revocations	Yes, with sig	Yes, with revocations	Yes, with revocations

#### 4. Sources of Bootstrapping Data

This section defines some sources for bootstrapping data that a device can access. The list of sources defined here is not meant to be exhaustive. It is left to future documents to define additional sources for obtaining bootstrapping data.

For each source of bootstrapping data defined in this section, details are given for how the three artifacts listed in Section 3 are provided.

#### 4.1. Removable Storage

A directly attached removable storage device (e.g., a USB flash drive) MAY be used as a source of SZTP bootstrapping data.

Use of a removable storage device is compelling, as it does not require any external infrastructure to work. It is notable that the raw boot image file can also be located on the removable storage device, enabling a removable storage device to be a fully self-standing bootstrapping solution.

To use a removable storage device as a source of bootstrapping data, a device need only detect if the removable storage device is plugged in and mount its filesystem.

A removable storage device is an untrusted source of bootstrapping data. This means that the information stored on the removable storage device either MUST be signed or MUST be information that can be processed provisionally (e.g., unsigned redirect information).

From an artifact perspective, since a removable storage device presents itself as a filesystem, the bootstrapping artifacts need to be presented as files. The three artifacts defined in Section 3 are mapped to files below.

##### Artifact to File Mapping:

Conveyed Information: Mapped to a file containing the binary artifact described in Section 3.1 (e.g., conveyed-information.cms).

Owner Certificate: Mapped to a file containing the binary artifact described in Section 3.2 (e.g., owner-certificate.cms).

Ownership Voucher: Mapped to a file containing the binary artifact described in Section 3.3 (e.g., ownership-voucher.cms or ownership-voucher.vcj).

The format of the removable storage device's filesystem and the naming of the files are outside the scope of this document. However, in order to facilitate interoperability, it is RECOMMENDED devices support open and/or standards based filesystems. It is also RECOMMENDED that devices assume a file naming convention that enables more than one instance of bootstrapping data (i.e., for different devices) to exist on a removable storage device. The file naming convention SHOULD additionally be unique to the manufacturer, in

order to enable bootstrapping data from multiple manufacturers to exist on a removable storage device.

#### 4.2. DNS Server

A DNS server MAY be used as a source of SZTP bootstrapping data.

Using a DNS server may be a compelling option for deployments having existing DNS infrastructure, as it enables a touchless bootstrapping option that does not entail utilizing an Internet based resource hosted by a 3rd-party.

DNS is an untrusted source of bootstrapping data. Even if DNSSEC [RFC6698] is used to authenticate the various DNS resource records (e.g., A, AAAA, CERT, TXT, and TLSA), the device cannot be sure that the domain returned to it from e.g., a DHCP server, belongs to its rightful owner. This means that the information stored in the DNS records either MUST be signed (per this document, not DNSSEC), or MUST be information that can be processed provisionally (e.g., unsigned redirect information).

##### 4.2.1. DNS Queries

Devices claiming to support DNS as a source of bootstrapping data MUST first query for device-specific DNS records and, only if doing so does not result in a successful bootstrap, then MUST query for device-independent DNS records.

For each of the device-specific and device-independent queries, devices MUST first query using multicast DNS [RFC6762] and, only if doing so does not result in a successful bootstrap, then MUST query again using unicast DNS [RFC1035] [RFC7766], assuming the address of a DNS server is known, such as it may be using techniques similar to those described in Section 11 of [RFC6763], which is referenced a few times in this document, even though this document does not itself use DNS-SD (RFC 6763 is identified herein as an Informative reference).

When querying for device-specific DNS records, devices MUST query for TXT records [RFC1035] under "<serial-number>.\_sztp", where <serial-number> is the device's serial number (the same value as in the device's secure device identity certificate), and "\_sztp" is the globally scoped DNS attribute registered by this document in Section 10.7.

Example device-specific DNS record queries:

```
TXT in <serial-number>._sztp.local. (multicast)
TXT in <serial-number>._sztp.<domain>. (unicast)
```

When querying for device-independent DNS records, devices MUST query for SRV records [RFC2782] under "\_sztp.\_tcp", where "\_sztp" is the service name registered by this document in Section 10.6, and "\_tcp" is the globally scoped DNS attribute registered by [I-D.ietf-dnsop-attrleaf].

Note that a device-independent response is anyway only able to encode unsigned data, since signed data necessitates the use of a device-specific ownership voucher. Use of SRV records maximally leverages existing DNS standards. A response containing multiple SRV records is comparable to an unsigned redirect information's list of bootstrap servers.

Example device-independent DNS record queries:

```
SRV in _sztp._tcp.local. (multicast)
SRV in _sztp._tcp.<domain>. (unicast)
```

#### 4.2.2. DNS Response for Device-Specific Queries

For device-specific queries, the three bootstrapping artifacts defined in Section 3 are encoded into the TXT records using key/value pairs, similar to the technique described in Section 6.3 of [RFC6763].

Artifact to TXT Record Mapping:

Conveyed Information: Mapped to a TXT record having the key "ci" and the value being the binary artifact described in Section 3.1.

Owner Certificate: Mapped to a TXT record having the key "oc" and the value being the binary artifact described in Section 3.2.

Ownership Voucher: Mapped to a TXT record having the key "ov" and the value being the binary artifact described in Section 3.3.

Devices MUST ignore any other keys that may be returned.

Note that, despite the name, TXT records can and SHOULD (per Section 6.5 of [RFC6763]) encode binary data.

Following is an example of a device-specific response, as it might be presented by a user-agent, containing signed data. This example assumes that the device's serial number is "<serial-number>", the domain is "example.com", and that "<binary data>" represents the binary artifact:

```
<serial-number>._sztp.example.com. 3600 IN TXT "ci=<binary data>"  
<serial-number>._sztp.example.com. 3600 IN TXT "oc=<binary data>"  
<serial-number>._sztp.example.com. 3600 IN TXT "ov=<binary data>"
```

Note that, in the case that "ci" encodes unsigned data, the "oc" and "ov" keys would not be present in the response.

#### 4.2.3. DNS Response for Device-Independent Queries

For device-independent queries, the three bootstrapping artifacts defined in Section 3 are encoded into the SVR records as follows.

Artifact to SRV Record Mapping:

Conveyed Information: This artifact is not supported directly. Instead, the essence of unsigned redirect information is mapped to SVR records per [RFC2782].

Owner Certificate: Not supported. Device-independent responses are never encode signed data, and hence there is no need for an owner certificate artifact.

Ownership Voucher: Not supported. Device-independent responses are never encode signed data, and hence there is no need for an ownership voucher artifact.

Following is an example of a device-independent response, as it might be presented by a user-agent, containing (effectively) unsigned redirect information to four bootstrap servers. This example assumes that the domain is "example.com" and that there are four bootstrap servers "sztp[1-4]":

```
_sztp._tcp.example.com. 1800 IN SRV 0 0 443 sztp1.example.com.  
_sztp._tcp.example.com. 1800 IN SRV 1 0 443 sztp2.example.com.  
_sztp._tcp.example.com. 1800 IN SRV 2 0 443 sztp3.example.com.  
_sztp._tcp.example.com. 1800 IN SRV 2 0 443 sztp4.example.com.
```

Note that, in this example, "sztp3" and "sztp4" have equal priority, and hence effectively represent a clustered pair of bootstrap servers. While "sztp1" and "sztp2" only have a single SRV record each, it may be that the record points to a load-balancer fronting a cluster of bootstrap servers.

While this document does not use DNS-SD [RFC6763], per Section 12.2 of that RFC, mDNS responses SHOULD also include all address records (type "A" and "AAAA") named in the SRV rdata.

#### 4.2.4. Size of Signed Data

The signed data artifacts are large by DNS conventions. In the smallest-footprint scenario, they are each a few kilobytes in size. However, onboarding information can easily be several kilobytes in size, and has the potential to be many kilobytes in size.

All resource records, including TXT records, have an upper size limit of 65535 bytes, since "RDLENGTH" is a 16-bit field (Section 3.2.1 in [RFC1035]). If it is ever desired to encode onboarding information that exceeds this limit, the DNS records returned should instead encode redirect information, to direct the device to a bootstrap server from which the onboarding information can be obtained.

Given the expected size of the TXT records, it is unlikely that signed data will fit into a UDP-based DNS packet, even with the EDNS(0) Extensions [RFC6891] enabled. Depending on content, signed data may also not fit into a multicast DNS packet, which bounds the size to 9000 bytes, per Section 17 in [RFC6762]. Thus it is expected that DNS Transport over TCP [RFC7766] will be required in order to return signed data.

#### 4.3. DHCP Server

A DHCP server MAY be used as a source of SZTP bootstrapping data.

Using a DHCP server may be a compelling option for deployments having existing DHCP infrastructure, as it enables a touchless bootstrapping option that does not entail utilizing an Internet based resource hosted by a 3rd-party.

A DHCP server is an untrusted source of bootstrapping data. Thus the information stored on the DHCP server either MUST be signed, or it MUST be information that can be processed provisionally (e.g., unsigned redirect information).

However, unlike other sources of bootstrapping data described in this document, the DHCP protocol (especially DHCP for IPv4) is very limited in the amount of data that can be conveyed, to the extent that signed data cannot be communicated. This means that only unsigned redirect information can be conveyed via DHCP.

Since the redirect information is unsigned, it SHOULD NOT include the optional trust anchor certificate, as it takes up space in the DHCP message, and the device would have to discard it anyway. For this reason, the DHCP options defined in Section 8 do not enable the trust anchor certificate to be encoded.



From an artifact perspective, the three artifacts defined in Section 3 are mapped to the DHCP fields specified in Section 8 as follows.

Artifact to DHCP Option Fields Mapping:

Conveyed Information: This artifact is not supported directly. Instead, the essence of unsigned redirect information is mapped to the DHCP options described in Section 8.

Owner Certificate: Not supported. There is not enough space in the DHCP packet to hold an owner certificate artifact.

Ownership Voucher: Not supported. There is not enough space in the DHCP packet to hold an ownership voucher artifact.

#### 4.4. Bootstrap Server

A bootstrap server MAY be used as a source of SZTP bootstrapping data. A bootstrap server is defined as a RESTCONF [RFC8040] server implementing the YANG module provided in Section 7.

Using a bootstrap server as a source of bootstrapping data is a compelling option as it MAY use transport-level security, obviating the need for signed data, which may be easier to deploy in some situations.

Unlike any other source of bootstrapping data described in this document, a bootstrap server is not only a source of data, but it can also receive data from devices using the YANG-defined "report-progress" RPC defined in the YANG module (Section 7.3). The "report-progress" RPC enables visibility into the bootstrapping process (e.g., warnings and errors), and provides potentially useful information upon completion (e.g., the device's SSH host-keys).

A bootstrap server may be a trusted or an untrusted source of bootstrapping data, depending on if the device learned about the bootstrap server's trust anchor from a trusted source. When a bootstrap server is trusted, the conveyed information returned from it MAY be signed. When the bootstrap server is untrusted, the conveyed information either MUST be signed or MUST be information that can be processed provisionally (e.g., unsigned redirect information).

From an artifact perspective, since a bootstrap server presents data conforming to a YANG data model, the bootstrapping artifacts need to be mapped to YANG nodes. The three artifacts defined in Section 3

are mapped to "output" nodes of the "get-bootstrapping-data" RPC defined in Section 7.3 below.

Artifact to Bootstrap Server Mapping:

Conveyed Information: Mapped to the "conveyed-information" leaf in the output of the "get-bootstrapping-data" RPC.

Owner Certificate: Mapped to the "owner-certificate" leaf in the output of the "get-bootstrapping-data" RPC.

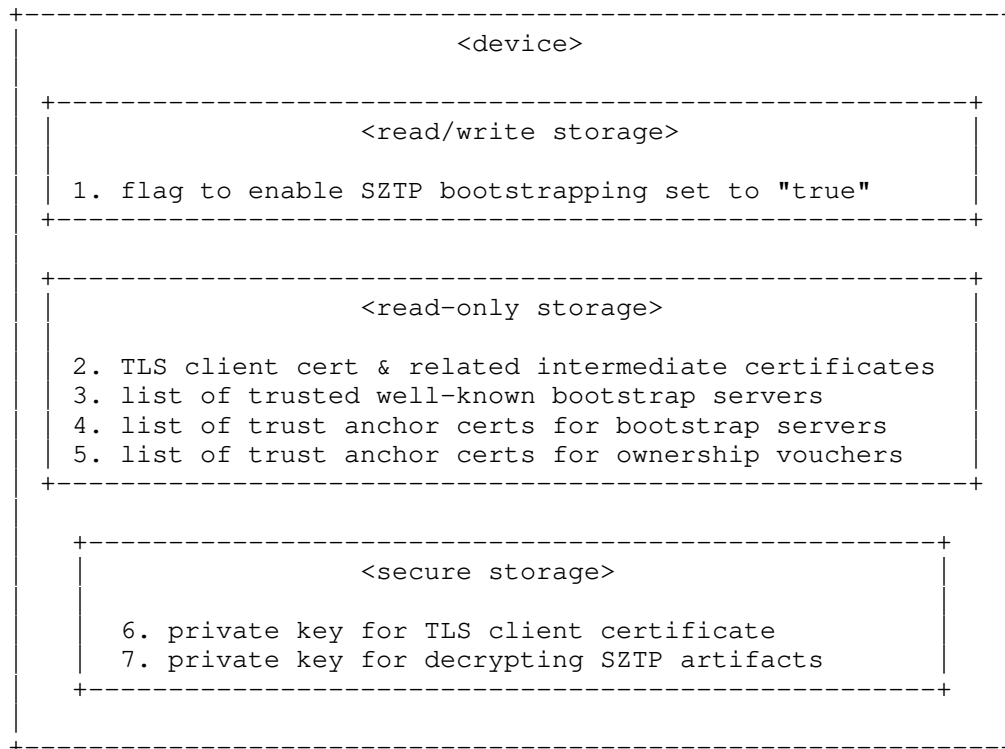
Ownership Voucher: Mapped to the "ownership-voucher" leaf in the output of the "get-bootstrapping-data" RPC.

SZTP bootstrap servers have only two endpoints, one for the "get-bootstrapping-data" RPC and one for the "report-progress" RPC. These RPCs use the authenticated RESTCONF username to isolate the execution of the RPC from other devices.

## 5. Device Details

Devices supporting the bootstrapping strategy described in this document MUST have the preconfigured state and bootstrapping logic described in the following sections.

### 5.1. Initial State



Each numbered item below corresponds to a numbered item in the diagram above.

1. Devices MUST have a configurable variable that is used to enable/disable SZTP bootstrapping. This variable MUST be enabled by default in order for SZTP bootstrapping to run when the device first powers on. Because it is a goal that the configuration installed by the bootstrapping process disables SZTP bootstrapping, and because the configuration may be merged into the existing configuration, using a configuration node that relies on presence is NOT RECOMMENDED, as it cannot be removed by the merging process.
2. Devices that support loading bootstrapping data from bootstrap servers (see Section 4.4) SHOULD possess a TLS-level client certificate and any intermediate certificates leading to the certificate's well-known trust-anchor. The well-known trust anchor certificate may be an intermediate certificate or a self-signed root certificate. To support devices not having a client certificate, devices MAY, alternatively or in addition to, identify and authenticate themselves to the bootstrap server

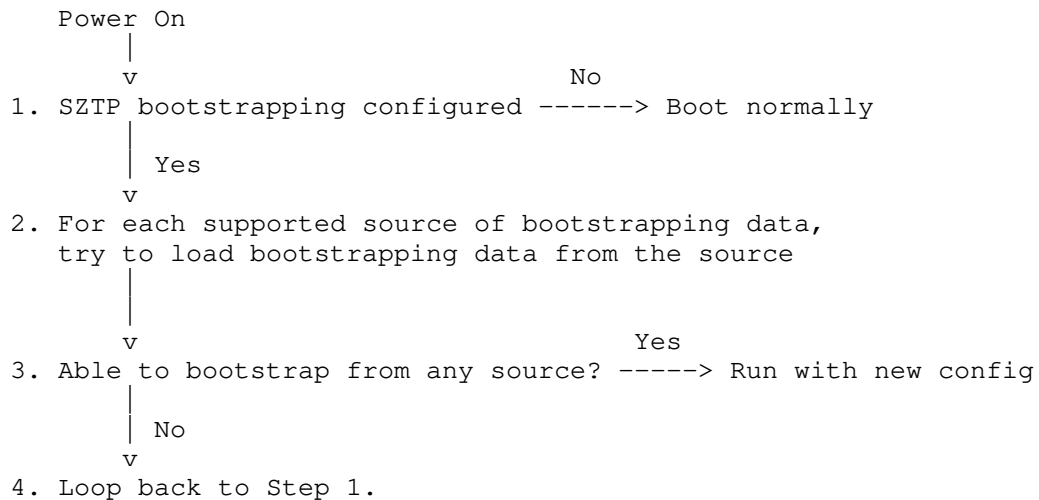
using an HTTP authentication scheme, as allowed by Section 2.5 in [RFC8040]; however, this document does not define a mechanism for operator input enabling, for example, the entering of a password.

3. Devices that support loading bootstrapping data from well-known bootstrap servers MUST possess a list of the well-known bootstrap servers. Consistent with redirect information (Section 2.1, each bootstrap server can be identified by its hostname or IP address, and an optional port.
4. Devices that support loading bootstrapping data from well-known bootstrap servers MUST also possess a list of trust anchor certificates that can be used to authenticate the well-known bootstrap servers. For each trust anchor certificate, if it is not itself a self-signed root certificate, the device SHOULD also possess the chain of intermediate certificates leading up to and including the self-signed root certificate.
5. Devices that support loading signed data (see Section 1.2) MUST possess the trust anchor certificates for validating ownership vouchers. For each trust anchor certificate, if it is not itself a self-signed root certificate, the device SHOULD also possess the chain of intermediate certificates leading up to and including the self-signed root certificate.
6. Devices that support using a TLS-level client certificate to identify and authenticate themselves to a bootstrap server MUST possess the private key that corresponds to the public key encoded in the TLS-level client certificate. This private key SHOULD be securely stored, ideally in a cryptographic processor, such as a trusted platform module (TPM) chip.
7. Devices that support decrypting SZTP artifacts MUST possess the private key that corresponds to the public key encoded in the secure device identity certificate used when encrypting the artifacts. This private key SHOULD be securely stored, ideally in a cryptographic processor, such as a trusted platform module (TPM) chip. This private key MAY be the same as the one associated to the TLS-level client certificate used when connecting to bootstrap servers.

A YANG module representing this data is provided in Appendix A.

## 5.2. Boot Sequence

A device claiming to support the bootstrapping strategy defined in this document MUST support the boot sequence described in this section.



Note: At any time, the device MAY be configured via an alternate provisioning mechanism (e.g., CLI).

Each numbered item below corresponds to a numbered item in the diagram above.

1. When the device powers on, it first checks to see if SZTP bootstrapping is configured, as is expected to be the case for the device's preconfigured initial state. If SZTP bootstrapping is not configured, then the device boots normally.
2. The device iterates over its list of sources for bootstrapping data (Section 4). Details for how to process a source of bootstrapping data are provided in Section 5.3.
3. If the device is able to bootstrap itself from any of the sources of bootstrapping data, it runs with the new bootstrapped configuration.
4. Otherwise the device MUST loop back through the list of bootstrapping sources again.

This document does not limit the simultaneous use of alternate provisioning mechanisms. Such mechanisms may include, for instance, a command line interface (CLI), a web-based user interface, or even another bootstrapping protocol. Regardless how it is configured, the configuration SHOULD unset the flag enabling SZTP bootstrapping discussed in Section 5.1.

### 5.3. Processing a Source of Bootstrapping Data

This section describes a recursive algorithm that devices can use to, ultimately, obtain onboarding information. The algorithm is recursive because sources of bootstrapping data may return redirect information, which causes the algorithm to run again, for the newly discovered sources of bootstrapping data. An expression that captures all possible successful sequences of bootstrapping data is: zero or more redirect information responses, followed by one onboarding information response.

An important aspect of the algorithm is knowing when data needs to be signed or not. The following figure provides a summary of options:

Kind of Bootstrapping Data	Untrusted Source Can Provide?	Trusted Source Can Provide?
Unsigned Redirect Info	: Yes+	Yes
Signed Redirect Info	: Yes	Yes*
Unsigned Onboarding Info	: No	Yes
Signed Onboarding Info	: Yes	Yes*

The '+' above denotes that the source redirected to MUST return signed data, or more unsigned redirect information.

The '\*' above denotes that, while possible, it is generally unnecessary for a trusted source to return signed data.

The recursive algorithm uses a conceptual global-scoped variable called "trust-state". The trust-state variable is initialized to FALSE. The ultimate goal of this algorithm is for the device to process onboarding information (Section 2.2) while the trust-state variable is TRUE.

If the source of bootstrapping data (Section 4) is a bootstrap server (Section 4.4), and the device is able to authenticate the bootstrap server using X.509 certificate path validation ([RFC6125], Section 6) to one of the device's preconfigured trust anchors, or to a trust anchor that it learned from a previous step, then the device MUST set trust-state to TRUE.

When establishing a connection to a bootstrap server, whether trusted or untrusted, the device MUST identify and authenticate itself to the bootstrap server using a TLS-level client certificate and/or an HTTP authentication scheme, per Section 2.5 in [RFC8040]. If both authentication mechanisms are used, they MUST both identify the same serial number.

When sending a client certificate, the device MUST also send all of the intermediate certificates leading up to, and optionally including, the client certificate's well-known trust anchor certificate.

For any source of bootstrapping data (e.g., Section 4), if any artifact obtained is encrypted, the device MUST first decrypt it using the private key associated with the device certificate used to encrypt the artifact.

If the conveyed information artifact is signed, and the device is able to validate the signed data using the algorithm described in Section 5.4, then the device MUST set trust-state to TRUE; otherwise, if the device is unable to validate the signed data, the device MUST set trust-state to FALSE. Note, this is worded to cover the special case when signed data is returned even from a trusted source of bootstrapping data.

If the conveyed information artifact contains redirect information, the device MUST, within limits of how many recursive loops the device allows, process the redirect information as described in Section 5.5. Implementations MUST limit the maximum number of recursive redirects allowed; the maximum number of recursive redirects allowed SHOULD be no more than ten. This is the recursion step, it will cause the device to reenter this algorithm, but this time the data source will definitely be a bootstrap server, as redirect information is only able to redirect devices to bootstrap servers.

If the conveyed information artifact contains onboarding information, and trust-state is FALSE, the device MUST exit the recursive algorithm (as this is not allowed, see the figure above), returning to the bootstrapping sequence described in Section 5.2. Otherwise, the device MUST attempt to process the onboarding information as described in Section 5.6. Whether the processing of the onboarding information succeeds or fails, the device MUST exit the recursive algorithm, returning to the bootstrapping sequence described in Section 5.2, the only difference being in how it responds to the "Able to bootstrap from any source?" conditional described in the figure in the section.

#### 5.4. Validating Signed Data

Whenever a device is presented signed data, it MUST validate the signed data as described in this section. This includes the case where the signed data is provided by a trusted source.

Whenever there is signed data, the device MUST also be provided an ownership voucher and an owner certificate. How all the needed

artifacts are provided for each source of bootstrapping data is described in Section 4.

In order to validate signed data, the device MUST first authenticate the ownership voucher by validating its signature to one of its preconfigured trust anchors (see Section 5.1), which may entail using additional intermediate certificates attached to the ownership voucher. If the device has an accurate clock, it MUST verify that the ownership voucher was created in the past (i.e., "created-on" < now) and, if the "expires-on" leaf is present, the device MUST verify that the ownership voucher has not yet expired (i.e., now < "expires-on"). The device MUST verify that the ownership voucher's "assertion" value is acceptable (e.g., some devices may only accept the assertion value "verified"). The device MUST verify that the ownership voucher specifies the device's serial number in the "serial-number" leaf. If the "idevid-issuer" leaf is present, the device MUST verify that the value is set correctly. If the authentication of the ownership voucher is successful, the device extracts the "pinned-domain-cert" node, an X.509 certificate, that is needed to verify the owner certificate in the next step.

The device MUST next authenticate the owner certificate by performing X.509 certificate path verification to the trusted certificate extracted from the ownership voucher's "pinned-domain-cert" node. This verification may entail using additional intermediate certificates attached to the owner certificate artifact. If the ownership voucher's "domain-cert-revocation-checks" node's value is set to "true", the device MUST verify the revocation status of the certificate chain used to sign the owner certificate and, if suitably-fresh revocation status is unattainable or if it is determined that a certificate has been revoked, the device MUST NOT validate the owner certificate.

Finally, the device MUST verify that the conveyed information artifact was signed by the validated owner certificate.

If any of these steps fail, the device MUST invalidate the signed data and not perform any subsequent steps.

## 5.5. Processing Redirect Information

In order to process redirect information (Section 2.1), the device MUST follow the steps presented in this section.

Processing redirect information is straightforward; the device sequentially steps through the list of provided bootstrap servers until it can find one it can bootstrap from.



If a hostname is provided, and the hostname's DNS resolution is to more than one IP address, the device MUST attempt to connect to all of the DNS resolved addresses at least once, before moving on to the next bootstrap server. If the device is able to obtain bootstrapping data from any of the DNS resolved addresses, it MUST immediately process that data, without attempting to connect to any of the other DNS resolved addresses.

If the redirect information is trusted (e.g., trust-state is TRUE), and the bootstrap server entry contains a trust anchor certificate, then the device MUST authenticate the specified bootstrap server's TLS server certificate using X.509 certificate path validation ([RFC6125], Section 6) to the specified trust anchor. If the bootstrap server entry does not contain a trust anchor certificate device, the device MUST establish a provisional connection to the bootstrap server (i.e., by blindly accepting its server certificate), and set trust-state to FALSE.

If the redirect information is untrusted (e.g., trust-state is FALSE), the device MUST discard any trust anchors provided by the redirect information and establish a provisional connection to the bootstrap server (i.e., by blindly accepting its TLS server certificate).

#### 5.6. Processing Onboarding Information

In order to process onboarding information (Section 2.2), the device MUST follow the steps presented in this section.

When processing onboarding information, the device MUST first process the boot image information (if any), then execute the pre-configuration script (if any), then commit the initial configuration (if any), and then execute the post-configuration script (if any), in that order.

When the onboarding information is obtained from a trusted bootstrap server, the device MUST send the "bootstrap-initiated" progress report, and send either a terminating "boot-image-installed-rebooting", "bootstrap-complete", or error specific progress report. If the bootstrap server's "get-bootstrapping-data" RPC-reply's "reporting-level" node is set to "verbose", the device MUST additionally send all appropriate non-terminating progress reports (e.g., initiated, warning, complete, etc.). Regardless of the reporting-level indicated by the bootstrap server, the device MAY send progress reports beyond the mandatory ones specified for the given reporting level.

When the onboarding information is obtained from an untrusted bootstrap server, the device MUST NOT send any progress reports to the bootstrap server, even though the onboarding information was, necessarily, signed and authenticated. Please be aware that bootstrap servers are recommended to promote untrusted connections to trusted connections, in the last paragraph of Section 9.6, so as to, in part, be able to collect progress reports from devices.

If the device encounters an error at any step, it MUST stop processing the onboarding information and return to the bootstrapping sequence described in Section 5.2. In the context of a recursive algorithm, the device MUST return to the enclosing loop, not back to the very beginning. Some state MAY be retained from the bootstrapping process (e.g., updated boot image, logs, remnants from a script, etc.). However, the retained state MUST NOT be active in any way (e.g., no new configuration or running of software), and MUST NOT hinder the ability for the device to continue the bootstrapping sequence (i.e., process onboarding information from another bootstrap server).

At this point, the specific ordered sequence of actions the device MUST perform is described.

If the onboarding information is obtained from a trusted bootstrap server, the device MUST send a "bootstrap-initiated" progress report. It is an error if the device does not receive back the "204 No Content" HTTP status line. If an error occurs, the device MUST try to send a "bootstrap-error" progress report before exiting.

The device MUST parse the provided onboarding information document, to extract values used in subsequent steps. Whether using a stream-based parser or not, if there is an error when parsing the onboarding information, and the device is connected to a trusted bootstrap server, the device MUST try to send a "parsing-error" progress report before exiting.

If boot image criteria are specified, the device MUST first determine if the boot image it is running satisfies the specified boot image criteria. If the device is already running the specified boot image, then it skips the remainder of this step. If the device is not running the specified boot image, then it MUST download, verify, and install, in that order, the specified boot image, and then reboot. If connected to a trusted bootstrap server, the device MAY try to send a "boot-image-mismatch" progress report. To download the boot image, the device MUST only use the URIs supplied by the onboarding information. To verify the boot image, the device MUST either use one of the verification fingerprints supplied by the onboarding information, or use a cryptographic signature embedded into the boot

image itself using a mechanism not described by this document. Before rebooting, if connected to a trusted bootstrap server, the device MUST try to send a "boot-image-installed-rebooting" progress report. Upon rebooting, the bootstrapping process runs again, which will eventually come to this step again, but then the device will be running the specified boot image, and thus will move to processing the next step. If an error occurs at any step while the device is connected to a trusted bootstrap server (i.e., before the reboot), the device MUST try to send a "boot-image-error" progress report before exiting.

If a pre-configuration script has been specified, the device MUST execute the script, capture any output emitted from the script, and check if the script had any warnings or errors. If an error occurs while the device is connected to a trusted bootstrap server, the device MUST try to send a "pre-script-error" progress report before exiting.

If an initial configuration has been specified, the device MUST atomically commit the provided initial configuration, using the approach specified by the "configuration-handling" leaf. If an error occurs while the device is connected to a trusted bootstrap server, the device MUST try to send a "config-error" progress report before exiting.

If a post-configuration script has been specified, the device MUST execute the script, capture any output emitted from the script, and check if the script had any warnings or errors. If an error occurs while the device is connected to a trusted bootstrap server, the device MUST try to send a "post-script-error" progress report before exiting.

If the onboarding information was obtained from a trusted bootstrap server, and the result of the bootstrapping process did not disable the "flag to enable SZTP bootstrapping" described in Section 5.1, the device SHOULD send an "bootstrap-warning" progress report.

If the onboarding information was obtained from a trusted bootstrap server, the device MUST send a "bootstrap-complete" progress report. It is an error if the device does not receive back the "204 No Content" HTTP status line. If an error occurs, the device MUST try to send a "bootstrap-error" progress report before exiting.

At this point, the device has completely processed the bootstrapping data.

The device is now running its initial configuration. Notably, if NETCONF Call Home or RESTCONF Call Home [RFC8071] is configured, the

device initiates trying to establish the call home connections at this time.

#### Implementation Notes:

Implementations may vary in how to ensure no unwanted state is retained when an error occurs.

Following are some guidelines for if the implementation chooses to undo previous steps:

- \* When an error occurs, the device must rollback the current step and any previous steps.
- \* Most steps are atomic. For example, the processing of a configuration is specified above as atomic, and the processing of scripts is similarly specified as atomic in the "ietf-sztp-conveyed-info" YANG module.
- \* In case the error occurs after the initial configuration was committed, the device must restore the configuration to the configuration that existed prior to the configuration being committed.
- \* In case the error occurs after a script had executed successfully, it may be helpful for the implementation to define scripts as being able to take a conceptual input parameter indicating that the script should remove its previously set state.

## 6. The Conveyed Information Data Model

This section defines a YANG 1.1 [RFC7950] module that is used to define the data model for the conveyed information artifact described in Section 3.1. This data model uses the "yang-data" extension statement defined in [RFC8040]. Examples illustrating this data model are provided in Section 6.2.

### 6.1. Data Model Overview

The following tree diagram provides an overview of the data model for the conveyed information artifact.

```
module: ietf-sztp-conveyed-info

yang-data conveyed-information:
  +-- (information-type)
  +--:(redirect-information)
  |   +-- redirect-information
  |   |   +-- bootstrap-server* [address]
  |   |   |   +-- address          inet:host
  |   |   |   +-- port?           inet:port-number
  |   |   |   +-- trust-anchor?   cms
  |   +--:(onboarding-information)
  |   |   +-- onboarding-information
  |   |   |   +-- boot-image
  |   |   |   |   +-- os-name?          string
  |   |   |   |   +-- os-version?       string
  |   |   |   |   +-- download-uri*     inet:uri
  |   |   |   |   +-- image-verification* [hash-algorithm]
  |   |   |   |   |   +-- hash-algorithm identityref
  |   |   |   |   |   +-- hash-value    yang:hex-string
  |   |   |   +-- configuration-handling? enumeration
  |   |   +-- pre-configuration-script? script
  |   +-- configuration? binary
  +-- post-configuration-script? script
```

## 6.2. Example Usage

The following example illustrates how redirect information (Section 2.1) can be encoded using JSON.

```
{
  "ietf-sztp-conveyed-info:redirect-information" : {
    "bootstrap-server" : [
      {
        "address" : "sztp1.example.com",
        "port" : 8443,
        "trust-anchor" : "base64encodedvalue=="
      },
      {
        "address" : "sztp2.example.com",
        "port" : 8443,
        "trust-anchor" : "base64encodedvalue=="
      },
      {
        "address" : "sztp3.example.com",
        "port" : 8443,
        "trust-anchor" : "base64encodedvalue=="
      }
    ]
  }
}
```

The following example illustrates how onboarding information (Section 2.2) can be encoded using JSON.

[Note: '\ ' line wrapping for formatting only]

```
{
  "ietf-sztp-conveyed-info:onboarding-information" : {
    "boot-image" : {
      "os-name" : "VendorOS",
      "os-version" : "17.2R1.6",
      "download-uri" : [ "http://some/path/to/raw/file" ],
      "image-verification" : [
        {
          "hash-algorithm" : "ietf-sztp-conveyed-info:sha-256",
          "hash-value" : "ba:ec:cf:a5:67:82:b4:10:77:c6:67:a6:22:ab:\
7d:50:04:a7:8b:8f:0e:db:02:8b:f4:75:55:fb:c1:13:b2:33"
        }
      ]
    },
    "configuration-handling" : "merge",
    "pre-configuration-script" : "base64encodedvalue==",
    "configuration" : "base64encodedvalue==",
    "post-configuration-script" : "base64encodedvalue=="
  }
}
```

### 6.3. YANG Module

The conveyed information data model is defined by the YANG module presented in this section.

This module uses data types defined in [RFC5280], [RFC5652], [RFC6234], and [RFC6991], an extension statement from [RFC8040], and an encoding defined in [ITU.X690.2015].

```
<CODE BEGINS> file "ietf-sztp-conveyed-info@2019-01-15.yang"
module ietf-sztp-conveyed-info {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-sztp-conveyed-info";
  prefix sztp-info;

  import ietf-yang-types {
    prefix yang;
    reference "RFC 6991: Common YANG Data Types";
  }
  import ietf-inet-types {
    prefix inet;
    reference "RFC 6991: Common YANG Data Types";
  }
  import ietf-restconf {
    prefix rc;
    reference "RFC 8040: RESTCONF Protocol";
  }

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

  contact
    "WG Web:  http://tools.ietf.org/wg/netconf
    WG List:  <mailto:netconf@ietf.org>
    Author:   Kent Watsen <mailto:kwatsen@juniper.net>";

  description
    "This module defines the data model for the Conveyed
    Information artifact defined in RFC XXXX: Secure Zero Touch
    Provisioning (SZTP).

    The key words '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 (RFC 2119,
    RFC 8174) when, and only when, they appear in all
    capitals, as shown here.
```

Copyright (c) 2019 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 2019-01-15 {
  description
    "Initial version";
  reference
    "RFC XXXX: Secure Zero Touch Provisioning (SZTP)";
}

// identities

identity hash-algorithm {
  description
    "A base identity for hash algorithm verification";
}

identity sha-256 {
  base "hash-algorithm";
  description "The SHA-256 algorithm.";
  reference "RFC 6234: US Secure Hash Algorithms.";
}

// typedefs

typedef cms {
  type binary;
  description
    "A ContentInfo structure, as specified in RFC 5652,
    encoded using ASN.1 distinguished encoding rules (DER),
    as specified in ITU-T X.690.";
  reference
    "RFC 5652:
    Cryptographic Message Syntax (CMS)
    ITU-T X.690:
    Information technology - ASN.1 encoding rules:
    Specification of Basic Encoding Rules (BER),
    Canonical Encoding Rules (CER) and Distinguished
    Encoding Rules (DER).";
}
```



```
}

// yang-data

rc:yang-data "conveyed-information" {
  choice information-type {
    mandatory true;
    description
      "This choice statement ensures the response contains
       redirect-information or onboarding-information.";
    container redirect-information {
      description
        "Redirect information is described in Section 2.1 in
         RFC XXXX. Its purpose is to redirect a device to
         another bootstrap server.";
      reference
        "RFC XXXX: Secure Zero Touch Provisioning (SZTP)";
      list bootstrap-server {
        key "address";
        min-elements 1;
        description
          "A bootstrap server entry.";
        leaf address {
          type inet:host;
          mandatory true;
          description
            "The IP address or hostname of the bootstrap server the
             device should redirect to.";
        }
        leaf port {
          type inet:port-number;
          default "443";
          description
            "The port number the bootstrap server listens on. If no
             port is specified, the IANA-assigned port for 'https'
             (443) is used.";
        }
      }
      leaf trust-anchor {
        type cms;
        description
          "A CMS structure that MUST contain the chain of
           X.509 certificates needed to authenticate the TLS
           certificate presented by this bootstrap server.

           The CMS MUST only contain a single chain of
           certificates. The bootstrap server MUST only
           authenticate to last intermediate CA certificate
           listed in the chain."
      }
    }
  }
}
```

In all cases, the chain MUST include a self-signed root certificate. In the case where the root certificate is itself the issuer of the bootstrap server's TLS certificate, only one certificate is present.

If needed by the device, this CMS structure MAY also contain suitably fresh revocation objects with which the device can verify the revocation status of the certificates.

This CMS encodes the degenerate form of the SignedData structure that is commonly used to disseminate X.509 certificates and revocation objects (RFC 5280).";

```
reference
  "RFC 5280:
    Internet X.509 Public Key Infrastructure Certificate
    and Certificate Revocation List (CRL) Profile.";
}
}
}
container onboarding-information {
  description
    "Onboarding information is described in Section 2.2 in
    RFC XXXX. Its purpose is to provide the device everything
    it needs to bootstrap itself.";
  reference
    "RFC XXXX: Secure Zero Touch Provisioning (SZTP)";
  container boot-image {
    description
      "Specifies criteria for the boot image the device MUST
      be running, as well as information enabling the device
      to install the required boot image.";
    leaf os-name {
      type string;
      description
        "The name of the operating system software the device
        MUST be running in order to not require a software
        image upgrade (ex. VendorOS).";
    }
    leaf os-version {
      type string;
      description
        "The version of the operating system software the
        device MUST be running in order to not require a
        software image upgrade (ex. 17.3R2.1).";
    }
    leaf-list download-uri {
```

```
type inet:uri;
ordered-by user;
description
  "An ordered list of URIs to where the same boot image
  file may be obtained. How the URI schemes (http, ftp,
  etc.) a device supports are known is vendor specific.
  If a secure scheme (e.g., https) is provided, a device
  MAY establish an untrusted connection to the remote
  server, by blindly accepting the server's end-entity
  certificate, to obtain the boot image.";
}
list image-verification {
  must '../download-uri' {
    description
      "Download URIs must be provided if an image is to
      be verified.";
  }
  key hash-algorithm;
  description
    "A list of hash values that a device can use to verify
    boot image files with.";
  leaf hash-algorithm {
    type identityref {
      base "hash-algorithm";
    }
    description
      "Identifies the hash algorithm used.";
  }
  leaf hash-value {
    type yang:hex-string;
    mandatory true;
    description
      "The hex-encoded value of the specified hash
      algorithm over the contents of the boot image
      file.";
  }
}
}
leaf configuration-handling {
  type enumeration {
    enum "merge" {
      description
        "Merge configuration into the running datastore.";
    }
    enum "replace" {
      description
        "Replace the existing running datastore with the
        passed configuration.";
    }
  }
}
```

```
    }
  }
  must '../configuration';
  description
    "This enumeration indicates how the server should process
    the provided configuration.";
}
leaf pre-configuration-script {
  type script;
  description
    "A script that, when present, is executed before the
    configuration has been processed.";
}
leaf configuration {
  type binary;
  must '../configuration-handling';
  description
    "Any configuration known to the device. The use of
    the 'binary' type enables e.g., XML-content to be
    embedded into a JSON document. The exact encoding
    of the content, as with the scripts, is vendor
    specific.";
}
leaf post-configuration-script {
  type script;
  description
    "A script that, when present, is executed after the
    configuration has been processed.";
}
}
}
```

```
typedef script {
  type binary;
  description
    "A device specific script that enables the execution of
    commands to perform actions not possible thru configuration
    alone.
```

No attempt is made to standardize the contents, running context, or programming language of the script, other than that it can indicate if any warnings or errors occurred and can emit output. The contents of the script are considered specific to the vendor, product line, and/or model of the device.

If the script execution indicates that an warning occurred,

then the device MUST assume that the script had a soft error that the script believes will not affect manageability.

If the script execution indicates that an error occurred, the device MUST assume the script had a hard error that the script believes will affect manageability. In this case, the script is required to gracefully exit, removing any state that might hinder the device's ability to continue the bootstrapping sequence (e.g., process onboarding information obtained from another bootstrap server).";

```
    }  
  }  
<CODE ENDS>
```

## 7. The SZTP Bootstrap Server API

This section defines the API for bootstrap servers. The API is defined as that produced by a RESTCONF [RFC8040] server that supports the YANG 1.1 [RFC7950] module defined in this section.

### 7.1. API Overview

The following tree diagram provides an overview for the bootstrap server RESTCONF API.

```
module: ietf-sztp-bootstrap-server
```

```
rpcs:
  +---x get-bootstrapping-data
  |   +---w input
  |   |   +---w signed-data-preferred?    empty
  |   |   +---w hw-model?                  string
  |   |   +---w os-name?                   string
  |   |   +---w os-version?                string
  |   |   +---w nonce?                     binary
  |   +---ro output
  |   |   +---ro reporting-level?          enumeration {onboarding-server}?
  |   |   +---ro conveyed-information      cms
  |   |   +---ro owner-certificate?        cms
  |   |   +---ro ownership-voucher?        cms
  +---x report-progress {onboarding-server}?
  |   +---w input
  |   |   +---w progress-type              enumeration
  |   |   +---w message?                   string
  |   |   +---w ssh-host-keys
  |   |   |   +---w ssh-host-key* []
  |   |   |   |   +---w algorithm          string
  |   |   |   |   +---w key-data          binary
  |   |   +---w trust-anchor-certs
  |   |   |   +---w trust-anchor-cert*    cms
```

## 7.2. Example Usage

This section presents three examples illustrating the bootstrap server's API. Two examples are provided for the "get-bootstrapping-data" RPC (once to an untrusted bootstrap server, and again to a trusted bootstrap server), and one example for the "report-progress" RPC.

The following example illustrates a device using the API to fetch its bootstrapping data from an untrusted bootstrap server. In this example, the device sends the "signed-data-preferred" input parameter and receives signed data in the response.

## REQUEST

[Note: '\ ' line wrapping for formatting only]

```
POST /restconf/operations/ietf-sztp-bootstrap-server:get-bootstrappi\
ng-data HTTP/1.1
HOST: example.com
Content-Type: application/yang.data+xml
```

```
<input
  xmlns="urn:ietf:params:xml:ns:yang:ietf-sztp-bootstrap-server">
  <signed-data-preferred/>
</input>
```

## RESPONSE

```
HTTP/1.1 200 OK
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+xml
```

```
<output
  xmlns="urn:ietf:params:xml:ns:yang:ietf-sztp-bootstrap-server">
  <conveyed-information>base64encodedvalue==</conveyed-information>
  <owner-certificate>base64encodedvalue==</owner-certificate>
  <ownership-voucher>base64encodedvalue==</ownership-voucher>
</output>
```

The following example illustrates a device using the API to fetch its bootstrapping data from a trusted bootstrap server. In this example, the device sends addition input parameters to the bootstrap server, which it may use when formulating its response to the device.

## REQUEST

[Note: '\ ' line wrapping for formatting only]

```
POST /restconf/operations/ietf-sztp-bootstrap-server:get-bootstrapi\
ng-data HTTP/1.1
HOST: example.com
Content-Type: application/yang.data+xml
```

```
<input
  xmlns="urn:ietf:params:xml:ns:yang:ietf-sztp-bootstrap-server">
  <hw-model>model-x</hw-model>
  <os-name>vendor-os</os-name>
  <os-version>17.3R2.1</os-version>
  <nonce>extralongbase64encodedvalue=</nonce>
</input>
```

## RESPONSE

```
HTTP/1.1 200 OK
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+xml
```

```
<output
  xmlns="urn:ietf:params:xml:ns:yang:ietf-sztp-bootstrap-server">
  <reporting-level>verbose</reporting-level>
  <conveyed-information>base64encodedvalue==</conveyed-information>
</output>
```

The following example illustrates a device using the API to post a progress report to a bootstrap server. Illustrated below is the "bootstrap-complete" message, but the device may send other progress reports to the server while bootstrapping. In this example, the device is sending both its SSH host keys and a TLS server certificate, which the bootstrap server may, for example, pass to an NMS, as discussed in Appendix C.3.



## REQUEST

[Note: '\ ' line wrapping for formatting only]

```
POST /restconf/operations/ietf-sztp-bootstrap-server:report-progress\
HTTP/1.1
HOST: example.com
Content-Type: application/yang.data+xml
```

```
<input
  xmlns="urn:ietf:params:xml:ns:yang:ietf-sztp-bootstrap-server">
  <progress-type>bootstrap-complete</progress-type>
  <message>example message</message>
  <ssh-host-keys>
    <ssh-host-key>
      <algorithm>ssh-rsa</algorithm>
      <key-data>base64encodedvalue==</key-data>
    </ssh-host-key>
    <ssh-host-key>
      <algorithm>rsa-sha2-256</algorithm>
      <key-data>base64encodedvalue==</key-data>
    </ssh-host-key>
  </ssh-host-keys>
  <trust-anchor-certs>
    <trust-anchor-cert>base64encodedvalue==</trust-anchor-cert>
  </trust-anchor-certs>
</input>
```

## RESPONSE

```
HTTP/1.1 204 No Content
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
```

### 7.3. YANG Module

The bootstrap server's device-facing API is normatively defined by the YANG module defined in this section.

This module uses data types defined in [RFC4253], [RFC5652], [RFC5280], [RFC6960], and [RFC8366], uses an encoding defined in [ITU.X690.2015], and makes a reference to [RFC4250] and [RFC6187].

```
<CODE BEGINS> file "ietf-sztp-bootstrap-server@2019-01-15.yang"
module ietf-sztp-bootstrap-server {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-sztp-bootstrap-server";
  prefix sztp-svr;
```

## organization

"IETF NETCONF (Network Configuration) Working Group";

## contact

"WG Web: <<http://tools.ietf.org/wg/netconf/>>

WG List: <<mailto:netconf@ietf.org>>

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

## description

"This module defines an interface for bootstrap servers, as defined by RFC XXXX: Secure Zero Touch Provisioning (SZTP).

The key words '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 (RFC 2119, RFC 8174) when, and only when, they appear in all capitals, as shown here.

Copyright (c) 2019 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 2019-01-15 {

description

"Initial version";

reference

"RFC XXXX: Secure Zero Touch Provisioning (SZTP)";

}

// features

feature redirect-server {

description

"The server supports being a 'redirect server'.";

}

feature onboarding-server {

description

"The server supports being an 'onboarding server'.";

}

```
// typedefs

typedef cms {
    type binary;
    description
        "A CMS structure, as specified in RFC 5652, encoded using
        ASN.1 distinguished encoding rules (DER), as specified in
        ITU-T X.690.";
    reference
        "RFC 5652:
        Cryptographic Message Syntax (CMS)
        ITU-T X.690:
        Information technology - ASN.1 encoding rules:
        Specification of Basic Encoding Rules (BER),
        Canonical Encoding Rules (CER) and Distinguished
        Encoding Rules (DER).";
}

// RPCs

rpc get-bootstrapping-data {
    description
        "This RPC enables a device, as identified by the RESTCONF
        username, to obtain bootstrapping data that has been made
        available for it.";
    input {
        leaf signed-data-preferred {
            type empty;
            description
                "This optional input parameter enables a device to
                communicate to the bootstrap server that it prefers
                to receive signed data. Devices SHOULD always send
                this parameter when the bootstrap server is untrusted.
                Upon receiving this input parameter, the bootstrap
                server MUST return either signed data, or unsigned
                redirect information; the bootstrap server MUST NOT
                return unsigned onboarding information.";
        }
        leaf hw-model {
            type string;
            description
                "This optional input parameter enables a device to
                communicate to the bootstrap server its vendor specific
                hardware model number. This parameter may be needed,
                for instance, when a device's IDevID certificate does
                not include the 'hardwareModelName' value in its
                subjectAltName field, as is allowed by 802.1AR-2009.";
            reference

```

```
        "IEEE 802.1AR-2009: IEEE Standard for Local and
          metropolitan area networks - Secure Device Identity";
    }
    leaf os-name {
        type string;
        description
            "This optional input parameter enables a device to
             communicate to the bootstrap server the name of its
             operating system. This parameter may be useful if
             the device, as identified by its serial number, can
             run more than one type of operating system (e.g.,
             on a white-box system.";
    }
    leaf os-version {
        type string;
        description
            "This optional input parameter enables a device to
             communicate to the bootstrap server the version of its
             operating system. This parameter may be used by a
             bootstrap server to return an operating system specific
             response to the device, thus negating the need for a
             potentially expensive boot-image update.";
    }
    leaf nonce {
        type binary {
            length "16..32";
        }
        description
            "This optional input parameter enables a device to
             communicate to the bootstrap server a nonce value.
             This may be especially useful for devices lacking
             an accurate clock, as then the bootstrap server
             can dynamically obtain from the manufacturer a
             voucher with the nonce value in it, as described
             in RFC 8366.";
        reference
            "RFC 8366:
             A Voucher Artifact for Bootstrapping Protocols";
    }
}
output {
    leaf reporting-level {
        if-feature onboarding-server;
        type enumeration {
            enum standard {
                description
                    "Send just the progress reports required by RFC XXXX.";
                reference

```

```
        "RFC XXXX: Secure Zero Touch Provisioning (SZTP)";
    }
    enum verbose {
        description
            "Send additional progress reports that might help
            troubleshooting an SZTP bootstrapping issue.";
    }
}
default standard;
description
    "Specifies the reporting level for progress reports the
    bootstrap server would like to receive when processing
    onboarding information. Progress reports are not sent
    when processing redirect information, or when the
    bootstrap server is untrusted (e.g., device sent the
    '<signed-data-preferred>' input parameter).";
}
leaf conveyed-information {
    type cms;
    mandatory true;
    description
        "An SZTP conveyed information artifact, as described in
        Section 3.1 of RFC XXXX.";
    reference
        "RFC XXXX: Secure Zero Touch Provisioning (SZTP)";
}
leaf owner-certificate {
    type cms;
    must '../ownership-voucher' {
        description
            "An ownership voucher must be present whenever an owner
            certificate is presented.";
    }
    description
        "An owner certificate artifact, as described in Section
        3.2 of RFC XXXX. This leaf is optional because it is
        only needed when the conveyed information artifact is
        signed.";
    reference
        "RFC XXXX: Secure Zero Touch Provisioning (SZTP)";
}
leaf ownership-voucher {
    type cms;
    must '../owner-certificate' {
        description
            "An owner certificate must be present whenever an
            ownership voucher is presented.";
    }
}
```

```
        description
            "An ownership voucher artifact, as described by Section
            3.3 of RFC XXXX. This leaf is optional because it is
            only needed when the conveyed information artifact is
            signed.";
        reference
            "RFC XXXX: Secure Zero Touch Provisioning (SZTP)";
    }
}

rpc report-progress {
    if-feature onboarding-server;
    description
        "This RPC enables a device, as identified by the RESTCONF
        username, to report its bootstrapping progress to the
        bootstrap server. This RPC is expected to be used when
        the device obtains onboarding-information from a trusted
        bootstrap server.";
    input {
        leaf progress-type {
            type enumeration {
                enum "bootstrap-initiated" {
                    description
                        "Indicates that the device just used the
                        'get-bootstrapping-data' RPC. The 'message' node
                        below MAY contain any additional information that
                        the manufacturer thinks might be useful.";
                }
                enum "parsing-initiated" {
                    description
                        "Indicates that the device is about to start parsing
                        the onboarding information. This progress type is
                        only for when parsing is implemented as a distinct
                        step.";
                }
                enum "parsing-warning" {
                    description
                        "Indicates that the device had a non-fatal error when
                        parsing the response from the bootstrap server. The
                        'message' node below SHOULD indicate the specific
                        warning that occurred.";
                }
                enum "parsing-error" {
                    description
                        "Indicates that the device encountered a fatal error
                        when parsing the response from the bootstrap server.
                        For instance, this could be due to malformed encoding,
```

```
the device expecting signed data when only unsigned
data is provided, the ownership voucher not listing
the device's serial number, or because the signature
didn't match. The 'message' node below SHOULD
indicate the specific error. This progress type
also indicates that the device has abandoned trying
to bootstrap off this bootstrap server.";
}
enum "parsing-complete" {
  description
    "Indicates that the device successfully completed
    parsing the onboarding information. This progress
    type is only for when parsing is implemented as a
    distinct step.";
}
enum "boot-image-initiated" {
  description
    "Indicates that the device is about to start
    processing the boot-image information.";
}
enum "boot-image-warning" {
  description
    "Indicates that the device encountered a non-fatal
    error condition when trying to install a boot-image.
    A possible reason might include a need to reformat a
    partition causing loss of data. The 'message' node
    below SHOULD indicate any warning messages that were
    generated.";
}
enum "boot-image-error" {
  description
    "Indicates that the device encountered an error when
    trying to install a boot-image, which could be for
    reasons such as a file server being unreachable,
    file not found, signature mismatch, etc. The
    'message' node SHOULD indicate the specific error
    that occurred. This progress type also indicates
    that the device has abandoned trying to bootstrap
    off this bootstrap server.";
}
enum "boot-image-mismatch" {
  description
    "Indicates that the device that has determined that
    it is not running the correct boot image. This
    message SHOULD precipitate trying to download
    a boot image.";
}
enum "boot-image-installed-rebooting" {
```

```
description
  "Indicates that the device successfully installed
  a new boot image and is about to reboot. After
  sending this progress type, the device is not
  expected to access the bootstrap server again
  for this bootstrapping attempt.";
}
enum "boot-image-complete" {
  description
    "Indicates that the device believes that it is
    running the correct boot-image.";
}
enum "pre-script-initiated" {
  description
    "Indicates that the device is about to execute the
    'pre-configuration-script'.";
}
enum "pre-script-warning" {
  description
    "Indicates that the device obtained a warning from the
    'pre-configuration-script' when it was executed. The
    'message' node below SHOULD capture any output the
    script produces.";
}
enum "pre-script-error" {
  description
    "Indicates that the device obtained an error from the
    'pre-configuration-script' when it was executed. The
    'message' node below SHOULD capture any output the
    script produces. This progress type also indicates
    that the device has abandoned trying to bootstrap
    off this bootstrap server.";
}
enum "pre-script-complete" {
  description
    "Indicates that the device successfully executed the
    'pre-configuration-script'.";
}
enum "config-initiated" {
  description
    "Indicates that the device is about to commit the
    initial configuration.";
}
enum "config-warning" {
  description
    "Indicates that the device obtained warning messages
    when it committed the initial configuration. The
    'message' node below SHOULD indicate any warning
```



```
        messages that were generated.";
    }
    enum "config-error" {
        description
            "Indicates that the device obtained error messages
            when it committed the initial configuration. The
            'message' node below SHOULD indicate the error
            messages that were generated. This progress type
            also indicates that the device has abandoned trying
            to bootstrap off this bootstrap server.";
    }
    enum "config-complete" {
        description
            "Indicates that the device successfully committed
            the initial configuration.";
    }
    enum "post-script-initiated" {
        description
            "Indicates that the device is about to execute the
            'post-configuration-script'.";
    }
    enum "post-script-warning" {
        description
            "Indicates that the device obtained a warning from the
            'post-configuration-script' when it was executed. The
            'message' node below SHOULD capture any output the
            script produces.";
    }
    enum "post-script-error" {
        description
            "Indicates that the device obtained an error from the
            'post-configuration-script' when it was executed. The
            'message' node below SHOULD capture any output the
            script produces. This progress type also indicates
            that the device has abandoned trying to bootstrap
            off this bootstrap server.";
    }
    enum "post-script-complete" {
        description
            "Indicates that the device successfully executed the
            'post-configuration-script'.";
    }
    enum "bootstrap-warning" {
        description
            "Indicates that a warning condition occurred for which
            there no other 'progress-type' enumeration is deemed
            suitable. The 'message' node below SHOULD describe
            the warning.";
```

```
    }
    enum "bootstrap-error" {
        description
            "Indicates that an error condition occurred for which
            there no other 'progress-type' enumeration is deemed
            suitable. The 'message' node below SHOULD describe
            the error. This progress type also indicates that
            the device has abandoned trying to bootstrap off
            this bootstrap server.";
    }
    enum "bootstrap-complete" {
        description
            "Indicates that the device successfully processed
            all 'onboarding-information' provided, and that it
            is ready to be managed. The 'message' node below
            MAY contain any additional information that the
            manufacturer thinks might be useful. After sending
            this progress type, the device is not expected to
            access the bootstrap server again.";
    }
    enum "informational" {
        description
            "Indicates any additional information not captured
            by any of the other progress types. For instance,
            a message indicating that the device is about to
            reboot after having installed a boot-image could
            be provided. The 'message' node below SHOULD
            contain information that the manufacturer thinks
            might be useful.";
    }
}
mandatory true;
description
    "The type of progress report provided.";
}
leaf message {
    type string;
    description
        "An optional arbitrary value.";
}
container ssh-host-keys {
    when "../progress-type = 'bootstrap-complete'" {
        description
            "SSH host keys are only sent when the progress type
            is 'bootstrap-complete'.";
    }
}
description
    "A list of SSH host keys an NMS may use to authenticate
```

```

        subsequent SSH-based connections to this device (e.g.,
        netconf-ssh, netconf-ch-ssh).";
list ssh-host-key {
  description
    "An SSH host key an NMS may use to authenticate
    subsequent SSH-based connections to this device
    (e.g., netconf-ssh, netconf-ch-ssh).";
  reference
    "RFC 4253: The Secure Shell (SSH) Transport Layer
    Protocol";
  leaf algorithm {
    type string;
    mandatory true;
    description
      "The public key algorithm name for this SSH key.

      Valid values are listed in the 'Public Key Algorithm
      Names' subregistry of the 'Secure Shell (SSH) Protocol
      Parameters' registry maintained by IANA.";
    reference
      "RFC 4250: The Secure Shell (SSH) Protocol Assigned
      Numbers
      IANA URL: https://www.iana.org/assignments/ssh-param\
eters/ssh-parameters.xhtml#ssh-parameters-19
      ('\\" added for formatting reasons)";
  }
  leaf key-data {
    type binary;
    mandatory true;
    description
      "The binary public key data for this SSH key, as
      specified by RFC 4253, Section 6.6, i.e.:

      string      certificate or public key format
                  identifier
      byte[n]     key/certificate data.";
    reference
      "RFC 4253: The Secure Shell (SSH) Transport Layer
      Protocol";
  }
}
}
}
container trust-anchor-certs {
  when "../progress-type = 'bootstrap-complete'" {
    description
      "Trust anchors are only sent when the progress type
      is 'bootstrap-complete'.";
  }
}

```

```
description
    "A list of trust anchor certificates an NMS may use to
    authenticate subsequent certificate-based connections
    to this device (e.g., restconf-tls, netconf-tls, or
    even netconf-ssh with X.509 support from RFC 6187).
    In practice, trust anchors for IDevID certificates do
    not need to be conveyed using this mechanism.";
reference
    "RFC 6187:
    X.509v3 Certificates for Secure Shell Authentication.";
leaf-list trust-anchor-cert {
    type cms;
    description
        "A CMS structure whose top-most content type MUST be the
        signed-data content type, as described by Section 5 in
        RFC 5652.

        The CMS MUST contain the chain of X.509 certificates
        needed to authenticate the certificate presented by
        the device.

        The CMS MUST contain only a single chain of
        certificates. The last certificate in the chain
        MUST be the issuer for the device's end-entity
        certificate.

        In all cases, the chain MUST include a self-signed
        root certificate. In the case where the root
        certificate is itself the issuer of the device's
        end-entity certificate, only one certificate is
        present.

        This CMS encodes the degenerate form of the SignedData
        structure that is commonly used to disseminate X.509
        certificates and revocation objects (RFC 5280).";
    reference
        "RFC 5280:
        Internet X.509 Public Key Infrastructure
        Certificate and Certificate Revocation List (CRL)
        Profile.
        RFC 5652:
        Cryptographic Message Syntax (CMS)";
}
}
}
}
}
<CODE ENDS>
```

## 8. DHCP Options

This section defines two DHCP options, one for DHCPv4 and one for DHCPv6. These two options are semantically the same, though syntactically different.

### 8.1. DHCPv4 SZTP Redirect Option

The DHCPv4 SZTP Redirect Option is used to provision the client with one or more URIs for bootstrap servers that can be contacted to attempt further configuration.

#### DHCPv4 SZTP Redirect Option

```

      0                                     1
      0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|  option-code (143)  |  option-length  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
.
.  bootstrap-server-list (variable length)  .
.
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

- \* option-code: OPTION\_V4\_SZTP\_REDIRECT (143)
- \* option-length: The option length in octets.
- \* bootstrap-server-list: A list of servers for the client to attempt contacting, in order to obtain further bootstrapping data, in the format shown in Section 8.3.

#### DHCPv4 Client Behavior

Clients MAY request the OPTION\_V4\_SZTP\_REDIRECT by including its option code in the Parameter Request List (55) in DHCP request messages.

On receipt of a DHCPv4 Reply message which contains the OPTION\_V4\_SZTP\_REDIRECT, the client processes the response according to Section 5.5, with the understanding that the "address" and "port" values are encoded in the URIs.

Any invalid URI entries received in the uri-data field are ignored by the client. If OPTION\_V4\_SZTP\_REDIRECT does not contain at least one valid URI entry in the uri-data field, then the client MUST discard the option.

As the list of URIs may exceed the maximum allowed length of a single DHCPv4 option (255 octets), the client MUST implement [RFC3396], allowing the URI list to be split across a number of OPTION\_V4\_SZTP\_REDIRECT option instances.

#### DHCPv4 Server Behavior

The DHCPv4 server MAY include a single instance of Option OPTION\_V4\_SZTP\_REDIRECT in DHCP messages it sends. Servers MUST NOT send more than one instance of the OPTION\_V4\_SZTP\_REDIRECT option.

The server's DHCP message MUST contain only a single instance of the OPTION\_V4\_SZTP\_REDIRECT's 'bootstrap-server-list' field. However, the list of URIs in this field may exceed the maximum allowed length of a single DHCPv4 option (per [RFC3396]).

If the length of 'bootstrap-server-list' is small enough to fit into a single instance of OPTION\_V4\_SZTP\_REDIRECT, the server MUST NOT send more than one instance of this option.

If the length of the 'bootstrap-server-list' field is too large to fit into a single option, then OPTION\_V4\_SZTP\_REDIRECT MUST be split into multiple instances of the option according to the process described in [RFC3396].

## 8.2. DHCPv6 SZTP Redirect Option

The DHCPv6 SZTP Redirect Option is used to provision the client with one or more URIs for bootstrap servers that can be contacted to attempt further configuration.

#### DHCPv6 SZTP Redirect Option

```

      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          option-code (136)          |          option-length          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
.          bootstrap-server-list (variable length)          .
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

- \* option-code: OPTION\_V6\_SZTP\_REDIRECT (136)
- \* option-length: The option length in octets.
- \* bootstrap-server-list: A list of servers for the client to attempt contacting, in order to obtain further bootstrapping data, in the format shown in Section 8.3.

#### DHCPv6 Client Behavior

Clients MAY request the `OPTION_V6_SZTP_REDIRECT` option, as defined in [RFC8415], Sections 18.2.1, 18.2.2, 18.2.4, 18.2.5, 18.2.6, and 21.7.

As a convenience to the reader, we mention here that the client includes requested option codes in the Option Request Option.

On receipt of a DHCPv6 Reply message which contains the `OPTION_V6_SZTP_REDIRECT`, the client processes the response according to Section 5.5, with the understanding that the "address" and "port" values are encoded in the URIs.

Any invalid URI entries received in the uri-data field are ignored by the client. If `OPTION_V6_SZTP_REDIRECT` does not contain at least one valid URI entry in the uri-data field, then the client MUST discard the option.

#### DHCPv6 Server Behavior

Section 18.3 of [RFC8415] governs server operation in regard to option assignment. As a convenience to the reader, we mention here that the server will send a particular option code only if configured with specific values for that option code and if the client requested it.

Option `OPTION_V6_SZTP_REDIRECT` is a singleton. Servers MUST NOT send more than one instance of the `OPTION_V6_SZTP_REDIRECT` option.

### 8.3. Common Field Encoding

Both of the DHCPv4 and DHCPv6 options defined in this section encode a list of bootstrap server URIs. The "URI" structure is a DHCP option that can contain multiple URIs (see [RFC7227], Section 5.7). Each URI entry in the bootstrap-server-list is structured as follows:

```

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+...+---+---+---+---+
|      uri-length      |      URI      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+...+---+---+---+---+

```

- \* uri-length: 2 octets long, specifies the length of the URI data.
- \* URI: URI of SZTP bootstrap server.

The URI of the SZTP bootstrap server MUST use the "https" URI scheme defined in Section 2.7.2 of [RFC7230], and MUST be in form "https://<ip-address-or-hostname>[:<port>]".

## 9. Security Considerations

### 9.1. Clock Sensitivity

The solution in this document relies on TLS certificates, owner certificates, and ownership vouchers, all of which require an accurate clock in order to be processed correctly (e.g., to test validity dates and revocation status). Implementations SHOULD ensure devices have an accurate clock when shipped from manufacturing facilities, and take steps to prevent clock tampering.

If it is not possible to ensure clock accuracy, it is RECOMMENDED that implementations disable the aspects of the solution having clock sensitivity. In particular, such implementations should assume that TLS certificates, ownership vouchers, and owner certificates never expire and are not revokable. From an ownership voucher perspective, manufacturers SHOULD issue a single ownership voucher for the lifetime of such devices.

Implementations SHOULD NOT rely on NTP for time, as NTP is not a secure protocol at this time. Note, there is an IETF work-in-progress to secure NTP [I-D.ietf-ntp-using-nts-for-ntp].

### 9.2. Use of IDevID Certificates

IDevID certificates, as defined in [Std-802.1AR-2018], are RECOMMENDED, both for the TLS-level client certificate used by devices when connecting to a bootstrap server, as well as for the device identity certificate used by owners when encrypting the SZTP bootstrapping data artifacts.

### 9.3. Immutable Storage for Trust Anchors

Devices MUST ensure that all their trust anchor certificates, including those for connecting to bootstrap servers and verifying ownership vouchers, 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.

### 9.4. Secure Storage for Long-lived Private Keys

Manufacturer-generated device identifiers may have very long lifetimes. For instance, [Std-802.1AR-2018] recommends using the "notAfter" value 99991231235959Z in IDevID certificates. Given the



long-lived nature of these private keys, it is paramount that they are stored so as to resist discovery, such as in a secure cryptographic processor, such as a trusted platform module (TPM) chip.

#### 9.5. Blindly Authenticating a Bootstrap Server

This document allows a device to blindly authenticate a bootstrap server's TLS certificate. It does so to allow for cases where the redirect information may be obtained in an unsecured manner, which is desirable to support in some cases.

To compensate for this, this document requires that devices, when connected to an untrusted bootstrap server, assert that data downloaded from the server is signed.

#### 9.6. Disclosing Information to Untrusted Servers

This document allows devices to establish connections to untrusted bootstrap servers. However, since the bootstrap server is untrusted, it may be under the control of an adversary, and therefore devices SHOULD be cautious about the data they send to the bootstrap server in such cases.

Devices send different data to bootstrap servers at each of the protocol layers TCP, TLS, HTTP, and RESTCONF.

At the TCP protocol layer, devices may relay their IP address, subject to network translations. Disclosure of this information is not considered a security risk.

At the TLS protocol layer, devices may use a client certificate to identify and authenticate themselves to untrusted bootstrap servers. At a minimum, the client certificate must disclose the device's serial number, and may disclose additional information such as the device's manufacturer, hardware model, public key, etc. Knowledge of this information may provide an adversary with details needed to launch an attack. It is RECOMMENDED that secrecy of the network constituency is not relied on for security.

At the HTTP protocol layer, devices may use an HTTP authentication scheme to identify and authenticate themselves to untrusted bootstrap servers. At a minimum, the authentication scheme must disclose the device's serial number and, concerningly, may, depending on the authentication mechanism used, reveal a secret that is only supposed to be known to the device (e.g., a password). Devices SHOULD NOT use an HTTP authentication scheme (e.g., HTTP Basic) with an untrusted

bootstrap server that reveals a secret that is only supposed to be known to the device.

At the RESTCONF protocol layer, devices use the "get-bootstrapping-data" RPC, but not the "report-progress" RPC, when connected to an untrusted bootstrap server. The "get-bootstrapping-data" RPC allows additional input parameters to be passed to the bootstrap server (e.g., "os-name", "os-version", "hw-model"). It is RECOMMENDED that devices only pass the "signed-data-preferred" input parameter to an untrusted bootstrap server. While it is okay for a bootstrap server to immediately return signed onboarding information, it is RECOMMENDED that bootstrap servers instead promote the untrusted connection to a trusted connection, as described in Appendix B, thus enabling the device to use the "report-progress" RPC while processing the onboarding information.

#### 9.7. Sequencing Sources of Bootstrapping Data

For devices supporting more than one source for bootstrapping data, no particular sequencing order has to be observed for security reasons, as the solution for each source is considered equally secure. However, from a privacy perspective, it is RECOMMENDED that devices access local sources before accessing remote sources.

#### 9.8. Safety of Private Keys used for Trust

The solution presented in this document enables bootstrapping data to be trusted in two ways, either through transport level security or through the signing of artifacts.

When transport level security (i.e., a trusted bootstrap server) is used, the private key for the end-entity certificate must be online in order to establish the TLS connection.

When artifacts are signed, the signing key is required to be online only when the bootstrap server is returning a dynamically generated signed-data response. For instance, a bootstrap server, upon receiving the "signed-data-preferred" input parameter to the "get-bootstrapping-data" RPC, may dynamically generate a response that is signed.

Bootstrap server administrators are RECOMMENDED to follow best practice to protect the private key used for any online operation. For instance, use of a hardware security module (HSM) is RECOMMENDED. If an HSM is not used, frequent private key refreshes are RECOMMENDED, assuming all bootstrapping devices have an accurate clock (see Section 9.1).

For best security, it is RECOMMENDED that owners only provide bootstrapping data that has been signed, using a protected private key, and encrypted, using the device's public key from its secure device identity certificate.

#### 9.9. Increased Reliance on Manufacturers

The SZTP bootstrapping protocol presented in this document shifts some control of initial configuration away from the rightful owner of the device and towards the manufacturer and its delegates.

The manufacturer maintains the list of well-known bootstrap servers its devices will trust. By design, if no bootstrapping data is found via other methods first, the device will try to reach out to the well-known bootstrap servers. There is no mechanism to prevent this from occurring other than by using an external firewall to block such connections. Concerns related to trusted bootstrap servers are discussed in Section 9.10.

Similarly, the manufacturer maintains the list of voucher signing authorities its devices will trust. The voucher signing authorities issue the vouchers that enable a device to trust an owner's domain certificate. It is vital that manufacturers ensure the integrity of these voucher signing authorities, so as to avoid incorrect assignments.

Operators should be aware that this system assumes that they trust all the pre-configured bootstrap servers and voucher signing authorities designated by the manufacturers. While operators may use points in the network to block access to the well-known bootstrap servers, operators cannot prevent voucher signing authorities from generating vouchers for their devices.

#### 9.10. Concerns with Trusted Bootstrap Servers

Trusted bootstrap servers, whether well-known or discovered, have the potential to cause problems, such as the following.

- o A trusted bootstrap server that has been compromised may be modified to return unsigned data of any sort. For instance, a bootstrap server that is only suppose to return redirect information might be modified to return onboarding information. Similarly, a bootstrap server that is only supposed to return signed data, may be modified to return unsigned data. In both cases, the device will accept the response, unaware that it wasn't supposed to be any different. It is RECOMMENDED that maintainers of trusted bootstrap servers ensure that their systems are not easily compromised and, in case of compromise, have mechanisms in

place to detect and remediate the compromise as expediently as possible.

- o A trusted bootstrap server hosting either unsigned, or signed but not encrypted, data may disclose information to unwanted parties (e.g., an administrator of the bootstrap server). This is a privacy issue only, but could reveal information that might be used in a subsequent attack. Disclosure of redirect information has limited exposure (it is just a list of bootstrap servers), whereas disclosure of onboarding information could be highly revealing (e.g., network topology, firewall policies, etc.). It is RECOMMENDED that operators encrypt the bootstrapping data when its contents are considered sensitive, even to the point of hiding it from the administrators of the bootstrap server, which may be maintained by a 3rd-party.

#### 9.11. Validity Period for Conveyed Information

The conveyed information artifact does not specify a validity period. For instance, neither redirect information nor onboarding information enable "not-before" or "not-after" values to be specified, and neither artifact alone can be revoked.

For unsigned data provided by an untrusted source of bootstrapping data, it is not meaningful to discuss its validity period when the information itself has no authenticity and may have come from anywhere.

For unsigned data provided by a trusted source of bootstrapping data (i.e., a bootstrap server), the availability of the data is the only measure of it being current. Since the untrusted data comes from a trusted source, its current availability is meaningful and, since bootstrap servers use TLS, the contents of the exchange cannot be modified or replayed.

For signed data, whether provided by an untrusted or trusted source of bootstrapping data, the validity is constrained by the validity of the both the ownership voucher and owner certificate used to authenticate it.

The ownership voucher's validity is primarily constrained by the ownership voucher's "created-on" and "expires-on" nodes. While [RFC8366] recommends short-lived vouchers (see Section 6.1), the "expires-on" node may be set to any point in the future, or omitted altogether to indicate that the voucher never expires. The ownership voucher's validity is secondarily constrained by the manufacturer's PKI used to sign the voucher; whilst an ownership voucher cannot be revoked directly, the PKI used to sign it may be.

The owner certificate's validity is primarily constrained by the X.509's validity field, the "notBefore" and "notAfter" values, as specified by the certificate authority that signed it. The owner certificate's validity is secondarily constrained by the validity of the PKI used to sign the voucher. Owner certificates may be revoked directly.

For owners that wish to have maximum flexibility in their ability to specify and constrain the validity of signed data, it is RECOMMENDED that a unique owner certificate is created for each signed artifact. Not only does this enable a validity period to be specified, for each artifact, but it also enables to the validity of each artifact to be revoked.

#### 9.12. Cascading Trust via Redirects

Redirect Information (Section 2.1), by design, instructs a bootstrapping device to initiate a HTTPS connection to the specified bootstrap servers.

When the redirect information is trusted, the redirect information can encode a trust anchor certificate used by the device to authenticate the TLS end-entity certificate presented by each bootstrap server.

As a result, any compromise in an interaction providing redirect information may result in compromise of all subsequent interactions.

#### 9.13. Possible Reuse of Private Keys

This document describes two uses for secure device identity certificates.

The primary use is for when the device authenticates itself to a bootstrap server, using its private key for TLS-level client-certificate based authentication.

A secondary use is for when the device needs to decrypt provided bootstrapping artifacts, using its private key to decrypt the data or, more precisely, per Section 6 in [RFC5652], decrypt a symmetric key used to decrypt the data.

This document, in Section 3.4 allows for the possibility that the same secure device identity certificate is used for both uses, as [Std-802.1AR-2018] states that a DevID certificate MAY have the "keyEncipherment" KeyUsage bit, in addition to the "digitalSignature" KeyUsage bit, set.

While it is understood that it is generally frowned upon to reuse private keys, this document views such reuse acceptable as there are not any known ways to cause a signature made in one context to be (mis)interpreted as valid in the other context.

#### 9.14. Non-Issue with Encrypting Signed Artifacts

This document specifies the encryption of signed objects, as opposed to the signing of encrypted objects, as might be expected given well-publicized oracle attacks (e.g., the padding oracle attack).

This document does not view such attacks as feasible in the context of the solution because the decrypted text never leaves the device.

#### 9.15. The "ietf-sztp-conveyed-info" YANG Module

The ietf-sztp-conveyed-info module defined in this document defines a data structure that is always wrapped by a CMS structure. When accessed by a secure mechanism (e.g., protected by TLS), then the CMS structure may be unsigned. However, when accessed by an insecure mechanism (e.g., removable storage device), then the CMS structure must be signed, in order for the device to trust it.

Implementations should be aware that signed bootstrapping data only protects the data from modification, and that the contents are still visible to others. This doesn't affect security so much as privacy. That the contents may be read by unintended parties when accessed by insecure mechanisms is considered next.

The ietf-sztp-conveyed-info module defines a top-level "choice" statement that declares the contents are either "redirect-information" or "onboarding-information". Each of these two cases are now considered.

When the content of the CMS structure is redirect-information, an observer can learn about the bootstrap servers the device is being directed to, their IP addresses or hostnames, ports, and trust anchor certificates. Knowledge of this information could provide an observer some insight into a network's inner structure.

When the content of the CMS structure is onboarding information, an observer could learn considerable information about how the device is to be provisioned. This information includes the operating system version, initial configuration, and script contents. This information should be considered sensitive and precautions should be taken to protect it (e.g., encrypt the artifact using the device's public key).

#### 9.16. The "ietf-sztp-bootstrap-server" YANG Module

The ietf-sztp-bootstrap-server module defined in this document specifies an API for a RESTCONF [RFC8040]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The NETCONF Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular users to a preconfigured subset of all available protocol operations and content.

This module presents no data nodes (only RPCs). There is no need to discuss the sensitivity of data nodes.

This module defines two RPC operations that may be considered sensitive in some network environments. These are the operations and their sensitivity/vulnerability:

**get-bootstrapping-data:** This RPC is used by devices to obtain their bootstrapping data. By design, each device, as identified by its authentication credentials (e.g. client certificate), can only obtain its own data. NACM is not needed to further constrain access to this RPC.

**report-progress:** This RPC is used by devices to report their bootstrapping progress. By design, each device, as identified by its authentication credentials (e.g. client certificate), can only report data for itself. NACM is not needed to further constrain access to this RPC.

### 10. IANA Considerations

#### 10.1. The IETF XML Registry

This document registers two URIs in the "ns" subregistry of the IETF XML Registry [RFC3688] maintained at <https://www.iana.org/assignments/xml-registry/xml-registry.xhtml#ns>. Following the format in [RFC3688], the following registrations are requested:

URI: urn:ietf:params:xml:ns:yang:ietf-sztp-conveyed-info  
Registrant Contact: The NETCONF WG of the IETF.  
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-sztp-bootstrap-server  
Registrant Contact: The NETCONF WG of the IETF.  
XML: N/A, the requested URI is an XML namespace.

## 10.2. The YANG Module Names Registry

This document registers two YANG modules in the YANG Module Names registry [RFC6020] maintained at <https://www.iana.org/assignments/yang-parameters/yang-parameters.xhtml>. Following the format defined in [RFC6020], the below registrations are requested:

```

name:      ietf-sztp-conveyed-info
namespace: urn:ietf:params:xml:ns:yang:ietf-sztp-conveyed-info
prefix:    sztp-info
reference:  RFC XXXX

name:      ietf-sztp-bootstrap-server
namespace: urn:ietf:params:xml:ns:yang:ietf-sztp-bootstrap-server
prefix:    sztp-svr
reference:  RFC XXXX

```

## 10.3. The SMI Security for S/MIME CMS Content Type Registry

This document registers two SMI security codes in the "SMI Security for S/MIME CMS Content Type" registry (1.2.840.113549.1.9.16.1) maintained at <https://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml#security-smime-1>. Following the format used in Section 3.4 of [RFC7107], the below registrations are requested:

Decimal	Description	References
-----	-----	-----
TBD1	id-ct-sztpConveyedInfoXML	[RFCXXXX]
TBD2	id-ct-sztpConveyedInfoJSON	[RFCXXXX]

id-ct-sztpConveyedInfoXML indicates that the "conveyed-information" is encoded using XML. id-ct-sztpConveyedInfoJSON indicates that the "conveyed-information" is encoded using JSON.

## 10.4. The BOOTP Manufacturer Extensions and DHCP Options Registry

This document registers one DHCP code point in the "BOOTP Manufacturer Extensions and DHCP Options" registry maintained at <http://www.iana.org/assignments/bootp-dhcp-parameters>. Following the format used by other registrations, the below registration is requested:

```

Tag:          143
Name:         OPTION_V4_SZTP_REDIRECT
Data Length:  N
Meaning:      This option provides a list of URIs
               for SZTP bootstrap servers
Reference:    [RFCXXXX]

```



Note: this request is to make permanent a previously registered early code point allocation.

#### 10.5. The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Registry

This document registers one DHCP code point in "Option Codes" subregistry of the "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)" registry maintained at <http://www.iana.org/assignments/dhcpv6-parameters>. Following the format used by other registrations, the below registration is requested:

Value:	136
Description:	OPTION_V6_SZTP_REDIRECT
Client ORO:	Yes
Singleton Option:	Yes
Reference:	[RFCXXXX]

Note: this request is to make permanent a previously registered early code point allocation.

#### 10.6. The Service Name and Transport Protocol Port Number Registry

This document registers one service name in the Service Name and Transport Protocol Port Number Registry [RFC6335] maintained at <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>. Following the format defined in Section 8.1.1 of [RFC6335], the below registration is requested:

Service Name:	sztp
Transport Protocol(s):	TCP
Assignee:	IESG <iesg@ietf.org>
Contact:	IETF Chair <chair@ietf.org>
Description:	This service name is used to construct the SRV service label "_sztp" for discovering SZTP bootstrap servers.
Reference:	[RFCXXXX]
Port Number:	N/A
Service Code:	N/A
Known Unauthorized Uses:	N/A
Assignment Notes:	This protocol uses HTTPS as a substrate.

#### 10.7. The DNS Underscore Global Scoped Entry Registry

This document registers one service name in the DNS Underscore Global Scoped Entry Registry [I-D.ietf-dnsop-attrleaf] maintained at TBD\_IANA\_URL. Following the format defined in Section 4.3 of [I-D.ietf-dnsop-attrleaf], the below registration is requested:

RR Type:                   TXT  
\_NODE NAME:               \_sztp  
Reference:                 [RFCXXXX]

## 11. References

### 11.1. Normative References

- [I-D.ietf-dnsop-attrleaf]  
Crocker, D., "DNS Scoped Data Through "Underscore" Naming of Attribute Leaves", draft-ietf-dnsop-attrleaf-16 (work in progress), November 2018.
- [ITU.X690.2015]  
International Telecommunication Union, "Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, ISO/IEC 8825-1, August 2015, <<https://www.itu.int/rec/T-REC-X.690/>>.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<https://www.rfc-editor.org/info/rfc1035>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <<https://www.rfc-editor.org/info/rfc2782>>.
- [RFC3396] Lemon, T. and S. Cheshire, "Encoding Long Options in the Dynamic Host Configuration Protocol (DHCPv4)", RFC 3396, DOI 10.17487/RFC3396, November 2002, <<https://www.rfc-editor.org/info/rfc3396>>.
- [RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <<https://www.rfc-editor.org/info/rfc4253>>.

- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <<https://www.rfc-editor.org/info/rfc5652>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <<https://www.rfc-editor.org/info/rfc6125>>.
- [RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <<https://www.rfc-editor.org/info/rfc6762>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC7227] Hankins, D., Mrugalski, T., Siodelski, M., Jiang, S., and S. Krishnan, "Guidelines for Creating New DHCPv6 Options", BCP 187, RFC 7227, DOI 10.17487/RFC7227, May 2014, <<https://www.rfc-editor.org/info/rfc7227>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.

- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <<https://www.rfc-editor.org/info/rfc8366>>.
- [RFC8415] Mrugalski, T., Siodelski, M., Volz, B., Yourtchenko, A., Richardson, M., Jiang, S., Lemon, T., and T. Winters, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 8415, DOI 10.17487/RFC8415, November 2018, <<https://www.rfc-editor.org/info/rfc8415>>.
- [Std-802.1AR-2018]  
IEEE SA-Standards Board, "IEEE Standard for Local and metropolitan area networks - Secure Device Identity", June 2018, <[https://standards.ieee.org/standard/802\\_1AR-2018.html](https://standards.ieee.org/standard/802_1AR-2018.html)>.

## 11.2. Informative References

- [I-D.ietf-netconf-crypto-types]  
Watsen, K. and H. Wang, "Common YANG Data Types for Cryptography", draft-ietf-netconf-crypto-types-02 (work in progress), October 2018.
- [I-D.ietf-netconf-trust-anchors]  
Watsen, K., "YANG Data Model for Global Trust Anchors", draft-ietf-netconf-trust-anchors-02 (work in progress), October 2018.
- [I-D.ietf-ntp-using-nts-for-ntp]  
Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network Time Protocol", draft-ietf-ntp-using-nts-for-ntp-15 (work in progress), December 2018.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC4250] Lehtinen, S. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Assigned Numbers", RFC 4250, DOI 10.17487/RFC4250, January 2006, <<https://www.rfc-editor.org/info/rfc4250>>.

- [RFC6187] Igoe, K. and D. Stebila, "X.509v3 Certificates for Secure Shell Authentication", RFC 6187, DOI 10.17487/RFC6187, March 2011, <<https://www.rfc-editor.org/info/rfc6187>>.
- [RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <<https://www.rfc-editor.org/info/rfc6234>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <<https://www.rfc-editor.org/info/rfc6335>>.
- [RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, DOI 10.17487/RFC6698, August 2012, <<https://www.rfc-editor.org/info/rfc6698>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<https://www.rfc-editor.org/info/rfc6763>>.
- [RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, DOI 10.17487/RFC6891, April 2013, <<https://www.rfc-editor.org/info/rfc6891>>.
- [RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <<https://www.rfc-editor.org/info/rfc6960>>.
- [RFC7107] Housley, R., "Object Identifier Registry for the S/MIME Mail Security Working Group", RFC 7107, DOI 10.17487/RFC7107, January 2014, <<https://www.rfc-editor.org/info/rfc7107>>.

- [RFC7766] Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., and D. Wessels, "DNS Transport over TCP - Implementation Requirements", RFC 7766, DOI 10.17487/RFC7766, March 2016, <<https://www.rfc-editor.org/info/rfc7766>>.
- [RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <<https://www.rfc-editor.org/info/rfc8071>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

## Appendix A. Example Device Data Model

This section defines a non-normative data model that enables the configuration of SZTP bootstrapping and discovery of what parameters are used by a device's bootstrapping logic.

### A.1. Data Model Overview

The following tree diagram provides an overview for the SZTP device data model.

```
module: example-device-data-model
  +--rw sztp
    +--rw enabled?                               boolean
    +--ro idevid-certificate?                     ct:end-entity-cert-cms
    | {bootstrap-servers}?
    +--ro bootstrap-servers {bootstrap-servers}?
    |   +--ro bootstrap-server* [address]
    |   |   +--ro address      inet:host
    |   |   +--ro port?       inet:port-number
    |   +--ro bootstrap-server-trust-anchors {bootstrap-servers}?
    |   |   +--ro reference*   ta:pinned-certificates-ref
    |   +--ro voucher-trust-anchors {signed-data}?
    |       +--ro reference*   ta:pinned-certificates-ref
```

In the above diagram, notice that there is only one configurable node "enabled". The expectation is that this node would be set to "true" in device's factory default configuration and that it would either be set to "false" or deleted when the SZTP bootstrapping is longer needed.

### A.2. Example Usage

Following is an instance example for this data model.

```
<sztp xmlns="https://example.com/sztp-device-data-model">
  <enabled>true</enabled>
  <idevid-certificate>base64encodedvalue==</idevid-certificate>
  <bootstrap-servers>
    <bootstrap-server>
      <address>sztp1.example.com</address>
      <port>8443</port>
    </bootstrap-server>
    <bootstrap-server>
      <address>sztp2.example.com</address>
      <port>8443</port>
    </bootstrap-server>
    <bootstrap-server>
      <address>sztp3.example.com</address>
      <port>8443</port>
    </bootstrap-server>
  </bootstrap-servers>
  <bootstrap-server-trust-anchors>
    <reference>manufacturers-root-ca-certs</reference>
  </bootstrap-server-trust-anchors>
  <voucher-trust-anchors>
    <reference>manufacturers-root-ca-certs</reference>
  </voucher-trust-anchors>
</sztp>
```

### A.3. YANG Module

The device model is defined by the YANG module defined in this section.

This module uses data types defined in [RFC6991], [I-D.ietf-netconf-crypto-types], and [I-D.ietf-netconf-trust-anchors].

```
module example-device-data-model {
  yang-version 1.1;
  namespace "https://example.com/sztp-device-data-model";
  prefix sztp-ddm;

  import ietf-inet-types {
    prefix inet;
    reference "RFC 6991: Common YANG Data Types";
  }

  import ietf-crypto-types {
    prefix ct;
    revision-date 2018-06-04;
    description
```



```
    "This revision is defined in the -00 version of
      draft-ietf-netconf-crypto-types";
  reference
    "draft-ietf-netconf-crypto-types:
      Common YANG Data Types for Cryptography";
}

import ietf-trust-anchors {
  prefix ta;
  revision-date 2018-06-04;
  description
    "This revision is defined in -00 version of
      draft-ietf-netconf-trust-anchors.";
  reference
    "draft-ietf-netconf-trust-anchors:
      YANG Data Model for Global Trust Anchors";
}

organization
  "Example Corporation";

contact
  "Author: Bootstrap Admin <mailto:admin@example.com>";

description
  "This module defines a data model to enable SZTP
    bootstrapping and discover what parameters are used.
    This module assumes the use of an IDevID certificate,
    as opposed to any other client certificate, or the
    use of an HTTP-based client authentication scheme.";

revision 2019-01-15 {
  description
    "Initial version";
  reference
    "RFC XXXX: Secure Zero Touch Provisioning (SZTP)";
}

// features

feature bootstrap-servers {
  description
    "The device supports bootstrapping off bootstrap servers.";
}

feature signed-data {
  description
    "The device supports bootstrapping off signed data.";
```

```
}

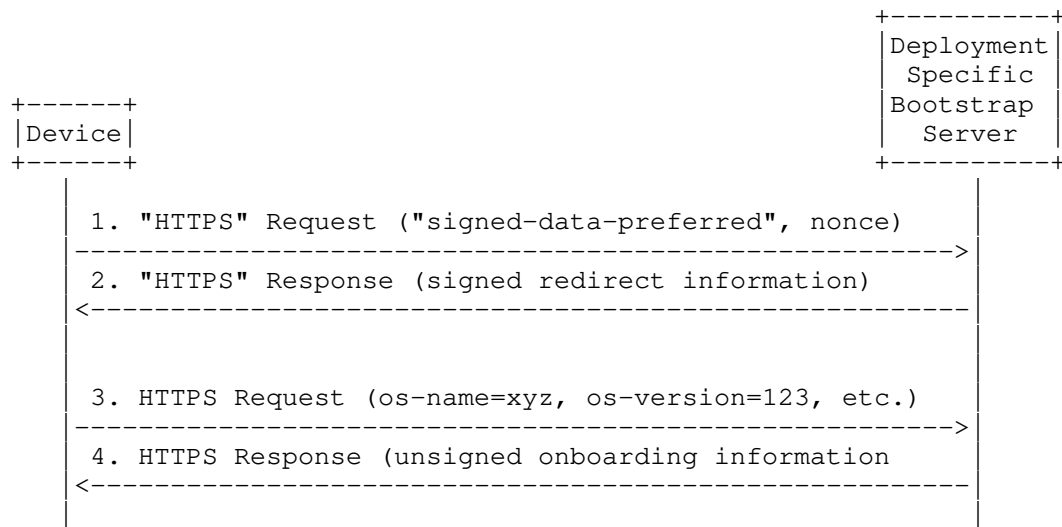
// protocol accessible nodes

container sztp {
  description
    "Top-level container for SZTP data model.";
  leaf enabled {
    type boolean;
    default false;
    description
      "The 'enabled' leaf controls if SZTP bootstrapping is
       enabled or disabled. The default is 'false' so that, when
       not enabled, which is most of the time, no configuration
       is needed.";
  }
  leaf idevid-certificate {
    if-feature bootstrap-servers;
    type ct:end-entity-cert-cms;
    config false;
    description
      "This CMS structure contains the IEEE 802.1AR-2009
       IDevID certificate itself, and all intermediate
       certificates leading up to, and optionally including,
       the manufacturer's well-known trust anchor certificate
       for IDevID certificates. The well-known trust anchor
       does not have to be a self-signed certificate.";
    reference
      "IEEE 802.1AR-2009:
       IEEE Standard for Local and metropolitan area
       networks - Secure Device Identity.";
  }
  container bootstrap-servers {
    if-feature bootstrap-servers;
    config false;
    description
      "List of bootstrap servers this device will attempt
       to reach out to when bootstrapping.";
    list bootstrap-server {
      key "address";
      description
        "A bootstrap server entry.";
      leaf address {
        type inet:host;
        mandatory true;
        description
          "The IP address or hostname of the bootstrap server the
           device should redirect to.";
      }
    }
  }
}
```

```
    }
    leaf port {
      type inet:port-number;
      default "443";
      description
        "The port number the bootstrap server listens on.  If no
        port is specified, the IANA-assigned port for 'https'
        (443) is used.";
    }
  }
}
container bootstrap-server-trust-anchors {
  if-feature bootstrap-servers;
  config false;
  description "Container for a list of trust anchor references.";
  leaf-list reference {
    type ta:pinned-certificates-ref;
    description
      "A reference to a list of pinned certificate authority (CA)
      certificates that the device uses to validate bootstrap
      servers with.";
  }
}
container voucher-trust-anchors {
  if-feature signed-data;
  config false;
  description "Container for a list of trust anchor references.";
  leaf-list reference {
    type ta:pinned-certificates-ref;
    description
      "A reference to a list of pinned certificate authority (CA)
      certificates that the device uses to validate ownership
      vouchers with.";
  }
}
}
```

## Appendix B. Promoting a Connection from Untrusted to Trusted

The following diagram illustrates a sequence of bootstrapping activities that promote an untrusted connection to a bootstrap server to a trusted connection to the same bootstrap server. This enables a device to limit the amount of information it might disclose to an adversary hosting an untrusted bootstrap server.



The interactions in the above diagram are described below.

1. The device initiates an untrusted connection to a bootstrap server, as is indicated by putting "HTTPS" in double quotes above. It is still an HTTPS connection, but the device is unable to authenticate the bootstrap server's TLS certificate. Because the device is unable to trust the bootstrap server, it sends the "signed-data-preferred" input parameter, and optionally also the "nonce" input parameter, in the "get-bootstrapping-data" RPC. The "signed-data-preferred" parameter informs the bootstrap server that the device does not trust it and may be holding back some additional input parameters from the server (e.g., other input parameters, progress reports, etc.). The "nonce" input parameter enables the bootstrap server to dynamically obtain an ownership voucher from a MASA, which may be important for devices that do not have a reliable clock.
2. The bootstrap server, seeing the "signed-data-preferred" input parameter, knows that it can either send unsigned redirect information or signed data of any type. But, in this case, the bootstrap server has the ability to sign data and chooses to respond with signed redirect information, not signed onboarding information as might be expected, securely redirecting the device back to it again. Not displayed but, if the "nonce" input parameter was passed, the bootstrap server could dynamically connect to a download a voucher from the MASA having the nonce value in it. Details regarding a protocol enabling this integration is outside the scope of this document.

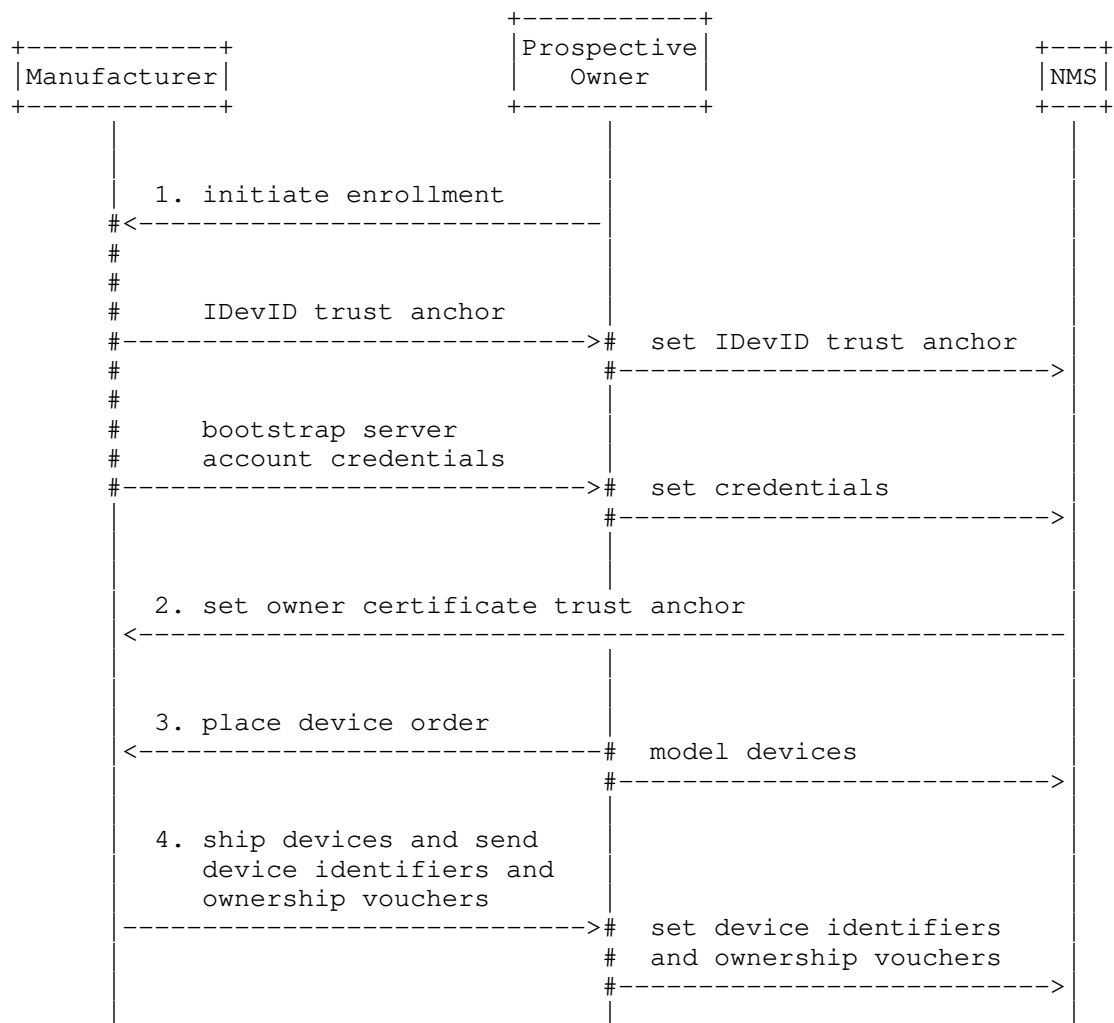
3. Upon validating the signed redirect information, the device establishes a secure connection to the bootstrap server. Unbeknownst to the device, it is the same bootstrap server it was connected to previously but, because the device is able to authenticate the bootstrap server this time, it sends its normal "get-bootstrapping-data" request (i.e., with additional input parameters) as well as its progress reports (not depicted).
4. This time, because the "signed-data-preferred" parameter was not passed, having access to all of the device's input parameters, the bootstrap server returns, in this example, unsigned onboarding information to the device. Note also that, because the bootstrap server is now trusted, the device will send progress reports to the server.

#### Appendix C. Workflow Overview

The solution presented in this document is conceptualized to be composed of the non-normative workflows described in this section. Implementation details are expected to vary. Each diagram is followed by a detailed description of the steps presented in the diagram, with further explanation on how implementations may vary.

##### C.1. Enrollment and Ordering Devices

The following diagram illustrates key interactions that may occur from when a prospective owner enrolls in a manufacturer's SZTP program to when the manufacturer ships devices for an order placed by the prospective owner.



Each numbered item below corresponds to a numbered item in the diagram above.

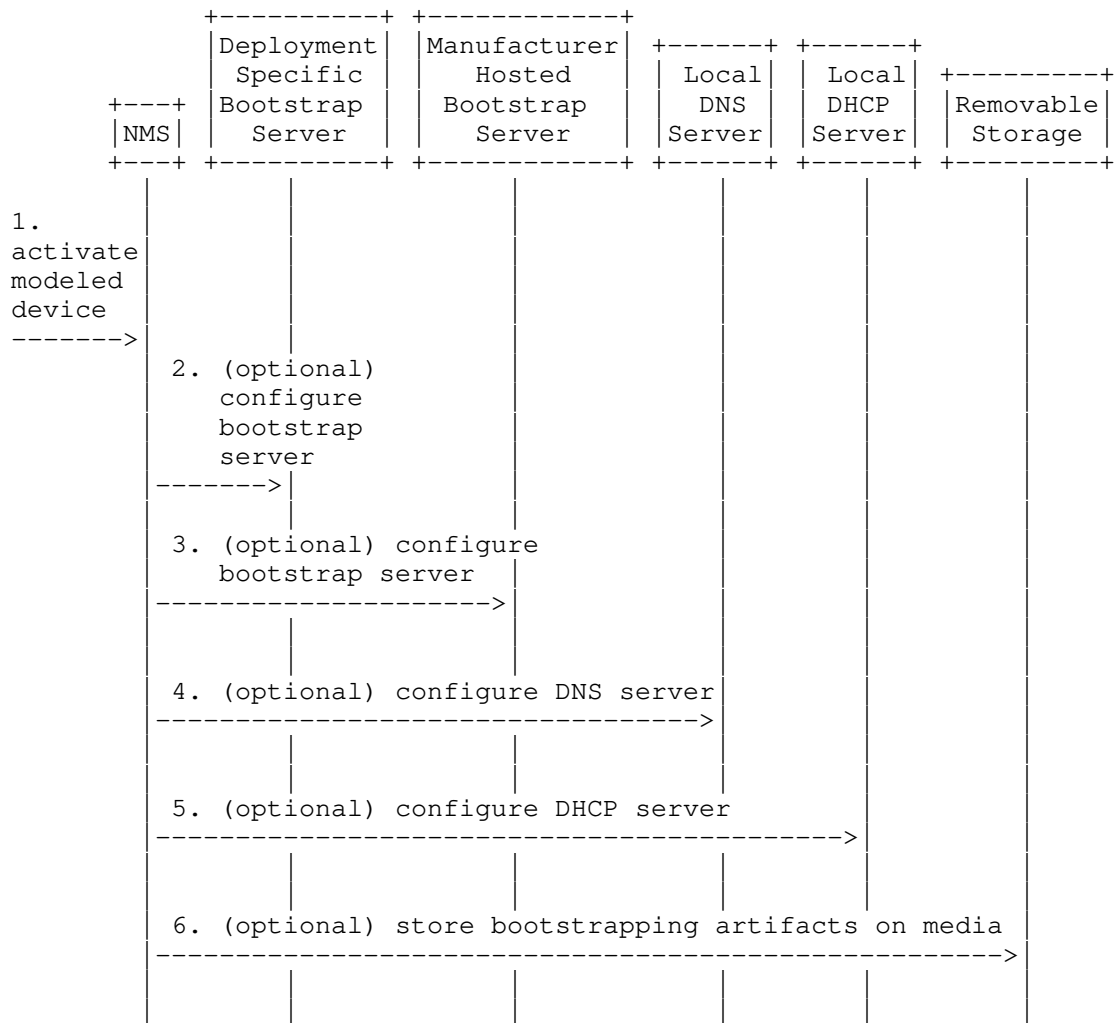
1. A prospective owner of a manufacturer's devices initiates an enrollment process with the manufacturer. This process includes the following:
  - \* Regardless how the prospective owner intends to bootstrap their devices, they will always obtain from the manufacturer the trust anchor certificate for the IDevID certificates. This certificate will be installed on the prospective owner's

NMS so that the NMS can authenticate the IDevID certificates when they are presented to subsequent steps.

- \* If the manufacturer hosts an Internet based bootstrap server (e.g., a redirect server) such as described in Section 4.4, then credentials necessary to configure the bootstrap server would be provided to the prospective owner. If the bootstrap server is configurable through an API (outside the scope of this document), then the credentials might be installed on the prospective owner's NMS so that the NMS can subsequently configure the manufacturer-hosted bootstrap server directly.
2. If the manufacturer's devices are able to validate signed data (Section 5.4), and assuming that the prospective owner's NMS is able to prepare and sign the bootstrapping data itself, the prospective owner's NMS might set a trust anchor certificate onto the manufacturer's bootstrap server, using the credentials provided in the previous step. This certificate is the trust anchor certificate that the prospective owner would like the manufacturer to place into the ownership vouchers it generates, thereby enabling devices to trust the owner's owner certificate. How this trust anchor certificate is used to enable devices to validate signed bootstrapping data is described in Section 5.4.
  3. Some time later, the prospective owner places an order with the manufacturer, perhaps with a special flag checked for SZTP handling. At this time, or perhaps before placing the order, the owner may model the devices in their NMS, creating 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 fulfills the order, shipping the devices to their intended locations, they may notify the owner of the devices' serial numbers and shipping destinations, which the owner may use to stage the network for when the devices power on. Additionally, the manufacturer may send one or more ownership vouchers, cryptographically assigning ownership of those devices to the owner. The owner may set this information on their NMS, perhaps binding specific modeled devices to the serial numbers and ownership vouchers.

#### C.2. Owner Stages the Network for Bootstrap

The following diagram illustrates how an owner might stage the network for bootstrapping devices.



Each numbered item below corresponds to a numbered item in the diagram above.

1. Having previously modeled the devices, including setting their fully operational configurations and associating device serial numbers and (optionally) ownership vouchers, the owner might "activate" one or more modeled devices. That is, the owner tells the NMS to perform the steps necessary to prepare for when the real-world devices power up and initiate the bootstrapping process. Note that, in some deployments, this step might be combined with the last step from the previous workflow. Here it

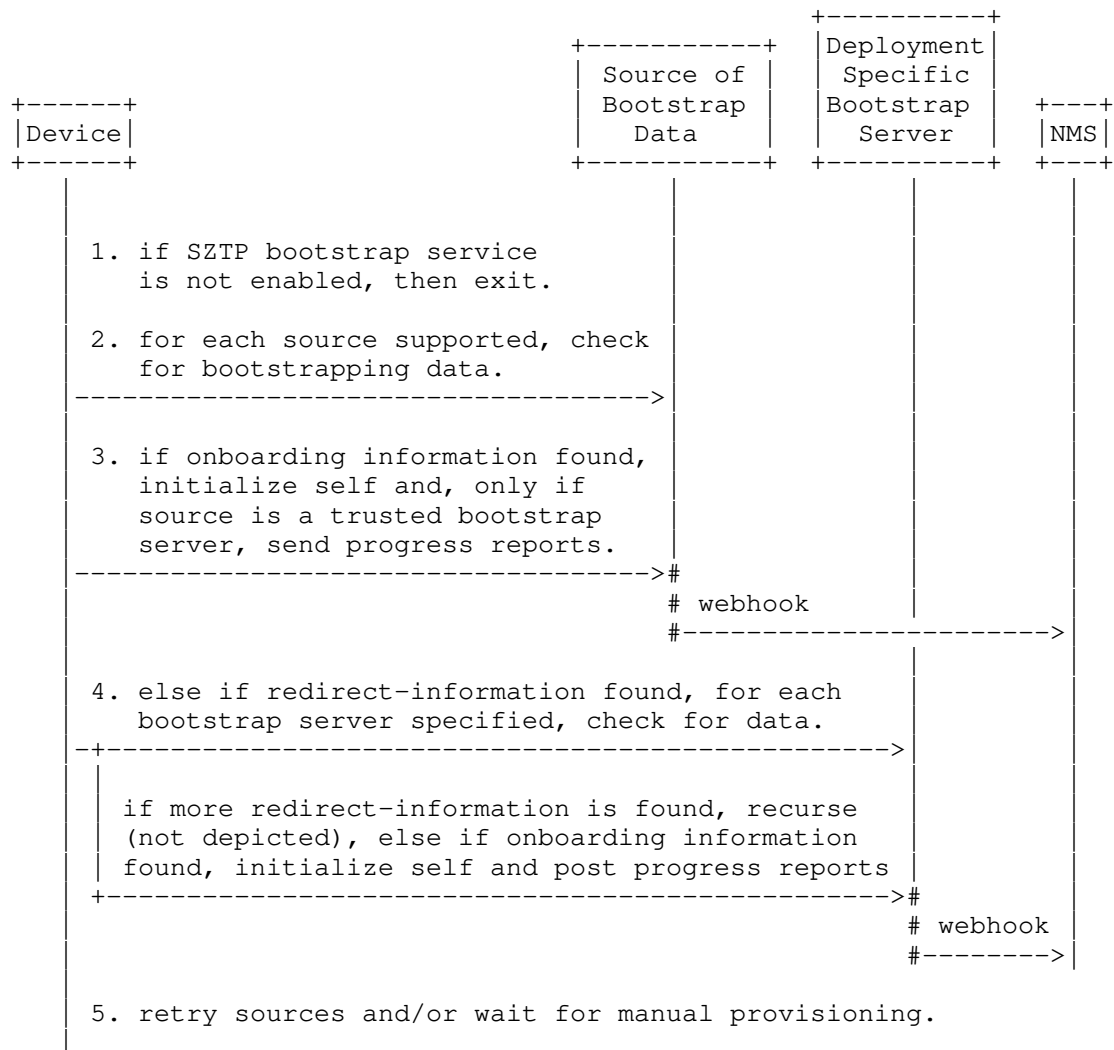


is depicted that an NMS performs the steps, but they may be performed manually or through some other mechanism.

2. If it is desired to use a deployment-specific bootstrap server, it must be configured to provide the bootstrapping data for the specific devices. Configuring the bootstrap server may occur via a programmatic API not defined by this document. Illustrated here as an external component, the bootstrap server may be implemented as an internal component of the NMS itself.
3. If it is desired to use a manufacturer hosted bootstrap server, it must be configured to provide the bootstrapping data for the specific devices. The configuration must be either redirect or onboarding information. That is, either the manufacturer hosted bootstrap server will redirect the device to another bootstrap server, or provide the device with the onboarding information itself. The types of bootstrapping data the manufacturer hosted bootstrap server supports may vary by implementation; some implementations may only support redirect information, or only support onboarding information, or support both redirect and onboarding information. Configuring the bootstrap server may occur via a programmatic API not defined by this document.
4. If it is desired to use a DNS server to supply bootstrapping data, a DNS server needs to be configured. If multicast DNS-SD is desired, then the DNS server must reside on the local network, otherwise the DNS server may reside on a remote network. Please see Section 4.2 for more information about how to configure DNS servers. Configuring the DNS server may occur via a programmatic API not defined by this document.
5. If it is desired to use a DHCP server to supply bootstrapping data, a DHCP server needs to be configured. The DHCP server may be accessed directly or via a DHCP relay. Please see Section 4.3 for more information about how to configure DHCP servers. Configuring the DHCP server may occur via a programmatic API not defined by this document.
6. If it is desired to use a removable storage device (e.g., USB flash drive) to supply bootstrapping data, the data would need to be placed onto it. Please see Section 4.1 for more information about how to configure a removable storage device.

### C.3. Device Powers On

The following diagram illustrates the sequence of activities that occur when a device powers on.



The interactions in the above diagram are described below.

1. Upon power being applied, the device checks to see if SZTP bootstrapping is configured, such as must be the case when running its "factory default" configuration. If SZTP bootstrapping is not configured, then the bootstrapping logic exits and none of the following interactions occur.
2. For each source of bootstrapping data the device supports, preferably in order of closeness to the device (e.g., removable

storage before Internet based servers), the device checks to see if there is any bootstrapping data for it there.

3. If onboarding information is found, the device initializes itself accordingly (e.g., installing a boot-image and committing an initial configuration). If the source is a bootstrap server, and the bootstrap server can be trusted (i.e., TLS-level authentication), the device also sends progress reports to the bootstrap server.

- \* The contents of the initial configuration should configure an administrator account on the device (e.g., username, SSH public key, etc.), and should configure the device either to listen for NETCONF or RESTCONF connections or to initiate call home connections [RFC8071], and should disable the SZTP bootstrapping service (e.g., the "enabled" leaf in data model presented in Appendix A).

- \* If the bootstrap server supports forwarding device progress reports to external systems (e.g., via a webhook), a "bootstrap-complete" progress report (Section 7.3) informs the external system to know when it can, for instance, initiate a connection to the device. To support this scenario further, the "bootstrap-complete" progress report may also relay the device's SSH host keys and/or TLS certificates, with which the external system can use to authenticate subsequent connections to the device.

If the device successfully completes the bootstrapping process, it exits the bootstrapping logic without considering any additional sources of bootstrapping data.

4. Otherwise, if redirect information is found, the device iterates through the list of specified bootstrap servers, checking to see if the bootstrap server has bootstrapping data for the device. If the bootstrap server returns more redirect information, then the device processes it recursively. Otherwise, if the bootstrap server returns onboarding information, the device processes it following the description provided in (3) above.
5. After having tried all supported sources of bootstrapping data, the device may retry again all the sources and/or provide manageability interfaces for manual configuration (e.g., CLI, HTTP, NETCONF, etc.). If manual configuration is allowed, and such configuration is provided, the configuration should also disable the SZTP bootstrapping service, as the need for bootstrapping would no longer be present.

## Appendix D. Change Log

## D.1. ID to 00

- o Major structural update; the essence is the same. Most every section was rewritten to some degree.
- o Added a Use Cases section
- o Added diagrams for "Actors and Roles" and "NMS Precondition" sections, and greatly improved the "Device Boot Sequence" diagram
- o Removed support for physical presence or any ability for configlets to not be signed.
- o Defined the Conveyed Information DHCP option
- o Added an ability for devices to also download images from configuration servers
- o Added an ability for configlets to be encrypted
- o Now configuration servers only have to support HTTP/S - no other schemes possible

## D.2. 00 to 01

- o Added boot-image and validate-owner annotations to the "Actors and Roles" diagram.
- o Fixed 2nd paragraph in section 7.1 to reflect current use of anyxml.
- o Added encrypted and signed-encrypted examples
- o Replaced YANG module with XSD schema
- o Added IANA request for the Conveyed Information DHCP Option
- o Added IANA request for media types for boot-image and configuration

## D.3. 01 to 02

- o 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.

- o Renamed configuration server to bootstrap server, a more representative name given the information devices download from it.
- o Replaced the concept of a configlet by defining a southbound interface for the bootstrap server using YANG.
- o Removed the IANA request for the boot-image and configuration media types

#### D.4. 02 to 03

- o Minor update, mostly just to add an Editor's Note to show how this draft might integrate with the draft-pritikin-anima-bootstrapping-keyinfra.

#### D.5. 03 to 04

- o Major update formally introducing unsigned data and support for Internet-based redirect servers.
- o Added many terms to Terminology section.
- o Added all new "Guiding Principles" section.
- o Added all new "Sources for Bootstrapping Data" section.
- o Rewrote the "Interactions" section and renamed it "Workflow Overview".

#### D.6. 04 to 05

- o Semi-major update, refactoring the document into more logical parts
- o Created new section for information types
- o Added support for DNS servers
- o Now allows provisional TLS connections
- o Bootstrapping data now supports scripts
- o Device Details section overhauled
- o Security Considerations expanded
- o Filled in enumerations for notification types

## D.7. 05 to 06

- o Minor update
- o Added many Normative and Informative references.
- o Added new section Other Considerations.

## D.8. 06 to 07

- o Minor update
- o Added an Editorial Note section for RFC Editor.
- o Updated the IANA Considerations section.

## D.9. 07 to 08

- o Minor update
- o Updated to reflect review from Michael Richardson.

## D.10. 08 to 09

- o Added in missing "Signature" artifact example.
- o Added recommendation for manufacturers to use interoperable formats and file naming conventions for removable storage devices.
- o Added configuration-handling leaf to guide if config should be merged, replaced, or processed like an edit-config/yang-patch document.
- o Added a pre-configuration script, in addition to the post-configuration script from -05 (issue #15).

## D.11. 09 to 10

- o Factored ownership voucher and voucher revocation to a separate document: draft-kwatsen-netconf-voucher. (issue #11)
- o Removed <configuration-handling> options "edit-config" and "yang-patch". (issue #12)
- o Defined how a signature over signed-data returned from a bootstrap server is processed. (issue #13)

- o Added recommendation for removable storage devices to use open/standard file systems when possible. (issue #14)
- o Replaced notifications "script-[warning/error]" with "[pre/post]-script-[warning/error]". (goes with issue #15)
- o switched owner-certificate to be encoded using the PKCS #7 format. (issue #16)
- o Replaced md5/sha1 with sha256 inside a choice statement, for future extensibility. (issue #17)
- o A ton of editorial changes, as I went thru the entire draft with a fine-toothed comb.

## D.12. 10 to 11

- o fixed yang validation issues found by IETFYANGPageCompilation. note: these issues were NOT found by pyang --ietf or by the submission-time validator...
- o fixed a typo in the yang module, someone the config false statement was removed.

## D.13. 11 to 12

- o fixed typo that prevented Appendix B from loading the examples correctly.
- o fixed more yang validation issues found by IETFYANGPageCompilation. note: again, these issues were NOT found by pyang --ietf or by the submission-time validator...
- o updated a few of the notification enumerations to be more consistent with the other enumerations (following the warning/error pattern).
- o updated the information-type artifact to state how it is encoded, matching the language that was in Appendix B.

## D.14. 12 to 13

- o defined a standalone artifact to encode the old information-type into a PKCS #7 structure.
- o standalone information artifact hardcodes JSON encoding (to match the voucher draft).

- o combined the information and signature PKCS #7 structures into a single PKCS #7 structure.
- o moved the certificate-revocations into the owner-certificate's PKCS #7 structure.
- o eliminated support for voucher-revocations, to reflect the voucher-draft's switch from revocations to renewals.

## D.15. 13 to 14

- o Renamed "bootstrap information" to "onboarding information".
- o Rewrote DHCP sections to address the packet-size limitation issue, as discussed in Chicago.
- o Added Ian as an author for his text-contributions to the DHCP sections.
- o Removed the Guiding Principles section.

## D.16. 14 to 15

- o Renamed action "notification" to "update-progress" and, likewise "notification-type" to "update-type".
- o Updated examples to use "base64encodedvalue==" for binary values.
- o Greatly simplified the "Artifact Groupings" section, and moved it as a subsection to the "Artifacts" section.
- o Moved the "Workflow Overview" section to the Appendix.
- o Renamed "bootstrap information" to "update information".
- o Removed "Other Considerations" section.
- o Tons of editorial updates.

## D.17. 15 to 16

- o tweaked language to refer to "initial state" rather than "factory default configuration", so as accommodate white-box scenarios.
- o added a paragraph to Intro regarding how the solution primarily regards physical machines, but could be extended to VMs by a future document.



- o added a pointer to the Workflow Overview section (recently moved to the Appendix) to the Intro.
- o added a note that, in order to simplify the verification process, the "Conveyed Information" PKCS #7 structure MUST also contain the signing X.509 certificate.
- o noted that the owner certificate's must either have no Key Usage or the Key Usage must set the "digitalSignature" bit.
- o noted that the owner certificate's subject and subjectAltName values are not constrained.
- o moved/consolidated some text from the Artifacts section down to the Device Details section.
- o tightened up some ambiguous language, for instance, by referring to specific leaf names in the Voucher artifact.
- o reverted a previously overzealous s/unique-id/serial-number/change.
- o modified language for when ZTP runs from when factory-default config is running to when ZTP is configured, which the factory-defaults should set .

D.18. 16 to 17

- o Added an example for how to promote an untrusted connection to a trusted connection.
- o Added a "query parameters" section defining some parameters enabling scenarios raised in last call.
- o Added a "Disclosing Information to Untrusted Servers" section to the Security Considerations.

D.19. 17 to 18

- o Added Security Considerations for each YANG module.
- o Reverted back to the device always sending its DevID cert.
- o Moved data tree to "get-bootstrapping-data" RPC.
- o Moved the "update-progress" action to a "report-progress" RPC.

- o Added an "signed-data-preferred" parameter to "get-bootstrapping-data" RPC.
- o Added the "ietf-zerotouch-device" module.
- o Lots of small updates.

#### D.20. 18 to 19

- o Fixed "must" expressions, by converting "choice" to a "list" of "image-verification", each of which now points to a base identity called "hash-algorithm". There's just one algorithm currently defined (sha-256). Wish there was a standard crypto module that could identify such identities.

#### D.21. 19 to 20

- o Now references I-D.ietf-netmod-yang-tree-diagrams.
- o Fixed tree-diagrams in Section 2 to always reflect current YANG (now they are now dynamically generated).
- o The "redirect-information" container's "trust-anchor" is now a CMS structure that can contain a chain of certificates, rather than a single certificate.
- o The "onboarding-information" container's support for image verification reworked to be extensible.
- o Added a reference to the "Device Details" section to the new example-device-data-model module.
- o Clarified that the device must always pass its IDevID certificate, even for untrusted bootstrap servers.
- o Fixed the description statement for the "script" typedef to refer to the [pre/post]-script-[warning/error] enums, rather than the legacy script-[warning/error] enums.
- o For the get-bootstrapping-data RPC's input, removed the "remote-id" and "circuit-id" fields, and added a "hw-model" field.
- o Improved DHCP error handling text.
- o Added MUST requirement for DHCPv6 client and server implementing [RFC3396] to handle URI lists longer than 255 octets.

- o Changed the "configuration" value in onboarding-information to be type "binary" instead of "anydata".
- o Moved everything from PKCS#7 to CMS (this shows up as a big change).
- o Added the early code point allocation assignments for the DHCP Options in the IANA Considerations section, and updated the RFC Editor note accordingly.
- o Added RFC Editor request to replace the assigned values for the CMS content types.
- o Relaxed auth requirements from device needing to always send IDevID cert to device needing to always send authentication credentials, as this better matches what RFC 8040 Section 2.5 says.
- o Moved normative module "ietf-zerotouch-device" to non-normative module "example-device-data-model".
- o Updated Title, Abstract, and Introduction per discussion on list.

## D.22. 20 to 21

- o Now any of the three artifact can be encrypted.
- o Fixed some line-too-long issues.

## D.23. 21 to 22

- o Removed specifics around how scripts indicate warnings or errors and how scripts emit output.
- o Moved the SZTP Device Data Model section to the Appendix.
- o Modified the YANG module in the SZTP Device Data Model section to reflect the latest trust-anchors and keystore drafts.
- o Modified types in other YANG modules to more closely emulate what is in draft-ietf-netconf-crypto-types.

## D.24. 22 to 23

- o Rewrote section 5.6 (processing onboarding information) to be clearer about error handling and retained state. Specifically:

- \* Clarified that a script, upon having an error, must gracefully exit, cleaning up any state that might hinder subsequent executions.
- \* Added ability for scripts to be executed again with a flag enabling them to clean up state from a previous execution.
- \* Clarified that the configuration commit is atomic.
- \* Clarified that any error encountered after committing the configuration (e.g., in the "post-configuration-script") must rollback the configuration to the previous configuration.
- \* Clarified that failure to successfully deliver the "bootstrap-initiated" and "bootstrap-complete" progress types must be treated as an error.
- \* Clarified that "return to bootstrapping sequence" is to be interpreted in the recursive context. Meaning that the device rolls-back one loop, rather than start over from scratch.
- o Changed how a device verifies a boot-image from just "MUST match one of the supplied fingerprints" to also allow for the verification to use an cryptographic signature embedded into the image itself.
- o Added more "progress-type" enums for visibility reasons, enabling more strongly-typed debug information to be sent to the bootstrap server.
- o Added Security Considerations based on early SecDir review.
- o Added recommendation for device to send warning if the initial config does not disable the bootstrapping process.

## D.25. 23 to 24

- o Follow-ups from SecDir and Shepherd.
- o Added "boot-image-complete" enumeration.

## D.26. 24 to 25

- o Removed remaining old "bootstrapping information" term usage.
- o Fixed DHCP Option length definition.
- o Added reference to RFC 6187.

## D.27. 25 to 26

- o Updated URI structure text (sec 8.3) and added norm. ref to RFC7230 reflecting Alexey Melnikov's comment.
- o Added IANA registration for the 'zerotouch' service, per IESG review from Adam Roach.
- o Clarified device's looping behavior and support for alternative provisioning mechanisms, per IESG review from Mirja Kuehlewind.
- o Updated "ietf-sztp-bootstrap-server:ssh-host-key" from leaf-list to list, per IESG review from Benjamin Kaduk.
- o Added option size text to DHCPv4 option size to address Suresh Krishnan's IESG review discuss point.
- o Updated RFC3315 to RFC8415 and associated section references.
- o Revamped the DNS Server section, after digging into Alexey Melnikov comment.
- o Fixed IETF terminology template section in both YANG modules.

## D.28. 26 to 27

- o Added Security Consideration for cascading trust via redirects.
- o Modified the get-bootstrapping-data RPC's "nonce" input parameter to being a minimum of 16-bytes (used to be 8-bytes).
- o Added Security Consideration regarding possible reuse of device's private key.
- o Added Security Consideration regarding use of sign-then-encrypt.
- o Renamed "Zero Touch"/"zerotouch" throughout. Now uses "SZTP" when referring to the draft/solution, and "conveyed" when referring to the bootstrapping artifact.
- o Added missing text for "encrypted unsigned conveyed information" case.
- o Renamed "untrusted-connection" input paramter to "signed-data-preferred"
- o Switch yd:yang-data back to rc:yang-data

- o Added a couple features to the bootstrap-server module.

#### D.29. 27 to 28

- o Modified DNS section to no longer reference DNS-SD (now just plain TXT and SRV lookups, via multicast or unicast).
- o Registers "\_sztp" in the DNS Underscore Global Scoped Entry Registry.
- o Updated 802.1AR reference to current spec version.

#### Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): Michael Behringer, Dean Bogdanovic, Martin Bjorklund, Joe Clarke, Dave Crocker, Toerless Eckert, Stephen Farrell, Stephen Hanna, Wes Hardaker, David Harrington, Mirja Kuehlewind, Radek Krejci, Suresh Krishnan, Benjamin Kaduk, David Mandelberg, Alexey Melnikov, Russ Mundy, Reinaldo Penno, Randy Presuhn, Max Pritikin, Michael Richardson, Adam Roach, Phil Shafer, Juergen Schoenwaelder.

Special thanks goes to Steve Hanna, Russ Mundy, and Wes Hardaker for brainstorming the original solution during the IETF 87 meeting in Berlin.

#### Authors' Addresses

Kent Watsen  
Juniper Networks

EMail: kwatsen@juniper.net

Mikael Abrahamsson  
T-Systems

EMail: mikael.abrahamsson@t-systems.se

Ian Farrer  
Deutsche Telekom AG

EMail: ian.farrer@telekom.de

netconf  
Internet-Draft  
Intended status: Standards Track  
Expires: April 30, 2015

B. Liu  
G. Zheng  
Huawei Technologies  
October 27, 2014

A NETCONF Extension for Data Fragmentation  
draft-liu-netconf-fragmentation-01

Abstract

This document introduces an extension to NETCONF (Network Configuration) protocol. The extension allows NETCONF to handle large size data as fragmented RPC messages. Specifically, this document defines a new <get-block> capability and relevant operations to handle the fragmentations.

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 30, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1. Introduction . . . . .	2
2. Requirements Language and Terminology . . . . .	3
3. Current Large size Handling Methods and Problems . . . . .	3
3.1. Stream-Oriented Handling . . . . .	3
3.2. Requesting a Portion of Data . . . . .	4
4. Candidate Solutions . . . . .	4
4.1. Netconf Fragmentation Mechanism . . . . .	4
4.1.1. Fragmentation Requirements . . . . .	4
4.1.2. <get-block> extention . . . . .	5
4.2. Subtree Iteration . . . . .	6
4.3. Linked Replies . . . . .	6
5. Security Considerations . . . . .	7
6. IANA Considerations . . . . .	7
7. Acknowledgements . . . . .	7
8. References . . . . .	7
8.1. Normative References . . . . .	7
8.2. Informative References . . . . .	7
Appendix A. Examples of the Candidate Solutions . . . . .	8
A.1. <get-next> (RPC Fragmentation) Example . . . . .	8
A.2. <get-list> (Subtree Iteration) Example . . . . .	10
A.3. Linked-replies Example . . . . .	11
Authors' Addresses . . . . .	12

## 1. Introduction

NETCONF [RFC6241] is the next generation network management protocol for configuring devices. It is becoming more and more popular, and some NMS (Network Management System) only use NETCONF as its southbound interface. The message procedures of NETCONF are based on RPC (Remote Procedure Call) interactions. A NETCONF client/server sends a <rpc> message to the counterpart and then receives a replying <rpc-reply> message.

In some situations, the <rpc-reply> message might be very large. For example, when NMS is retrieving a large amount of routes in a core router or doing a full-synchronizing with a device, the <rpc-reply> data might exceed Mega-Byte amount. Then there comes the problem of how to handle the large size data. In Section 3, this document briefly introduces two typical ways of current handling on this issue; and analyzes the problems of them.

To fix the problems, in Section 4, this document proposes a method of extending the NETCONF protocol to allow handling large size data as fragmented <rpc-reply> messages. The fragmentation is done at the NETCONF level, so it allows the NETCONF client to terminate the large size data processing momentarily by protocol interactions; and also



allows the fragmented messages to be instantly parsed piece by piece. Specifically, the fragmentation is achieved through a newly defined <get-block> capability and relevant operations.

## 2. Requirements Language and Terminology

The key words "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 [RFC2119] when they appear in ALL CAPS. When these words are not in ALL CAPS (such as "should" or "Should"), they have their usual English meanings, and are not to be interpreted as [RFC2119] key words.

Terminology:

DOM: Document Object Model, which is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents. Objects in the DOM tree may be addressed and manipulated by using methods on the objects. [DOM]

SAX: Simple API for XML, which is an event sequential access parser API developed by the XML-DEV mailing list for XML documents. SAX provides a mechanism for reading data from an XML document that is an alternative to that provided by the DOM. Where the DOM operates on the document as a whole, SAX parsers operate on each piece of the XML document sequentially. [SAX]

libxml: a software library for parsing XML documents. [LIBXML]

<get-block>: a capability and operation defined in this document to handle large size

## 3. Current Large size Handling Methods and Problems

### 3.1. Stream-Oriented Handling

Stream-Oriented handling mainly includes the following two aspects:

- o The server encapsulates the large size replying data in a <rpc-reply> message and streams it to the client through TCP protocol.
- o The client parses the received <rpc-reply> content in a stream-oriented way. More specifically, the client could utilize SAX [SAX-PARSING] to instantly parse the received content without waiting for the whole message been transported.

Problems:

- o Stream-Oriented method lacks the capability of discontinuing large size processing in the server. It would cause unnecessary resource/performance cost in the devices if the NMS has already got the intended portion or just canceled by the administrators.
- o Another problem is the implementation of SAX parsing is more complex than DOM parsing [DOM-PARSING] in the Netconf client. More computing burden will be taken in Netconf client to support SAX parsing.

### 3.2. Requesting a Portion of Data

The clients actively limit the search range of the data so that the servers only need to reply with a part of the large size data. Thus the clients could control the replies in a reasonable size. One example is that the clients get a list of the content, and provide a start offset and a max-count, to get a portion at a time.

Problems:

- o This method has an implication that the client needs to know the list/index of the intended large size data in advance before it starting the search request. It can't fit the scenarios of real-time on-demand data retrieving. And there is no standard to specify the list/index format in a uniform way. Thus it is only suitable for private implementation, thus multi-vendor interaction is not supported.
- o More important, it is just an indirect way to solve the problem. It could not fit the scenarios where the client just needs the whole large size data in the server.

## 4. Candidate Solutions

(Editor notes: this section discusses several possible solutions. The fragmentation mechanism is the original proposal of the draft. The other two were proposed during mailing list discussion by Andy Bierman and Juergen Schoenwaelder respectively. We include all of them for discussion and solution selection.)

### 4.1. Netconf Fragmentation Mechanism

#### 4.1.1. Fragmentation Requirements

this document proposes an RPC fragmentation mechanism to handle the large size data. Two essential requirements of the fragmentation are:

- o It needs to allow the NETCONF client to terminate the large size data processing momentarily by protocol interactions. In the proposed mechanisms in this draft, when the NETCONF server replies the client an <rpc-reply> fragmentation, it will wait the response from the client that whether it needs to send the next fragmentation. So if the initiator has got the intended portion, it could terminate the large size process immediately.
- o It needs to allow the NETCONF client to instantly parse the fragmentations piece by piece through the more widely supported DOM parsing. So in this document, it specifies that each <rpc-reply> fragmentation MUST be in a complete XML form.

#### 4.1.2. <get-block> extention

- o Function

The devices can only use <get-block> operation when the Get-block capability was announced.

The <get-block> fragmentation rules are:

- A. There should be a Max-Size for fragmentation. [Open Question]Should there be a clear specification of the size? E.g. 64K bytes.
- B. When the message reaches the Max-Size, it is sent to the client and the next message could be created in advance.
- C. Different records from one same table could be put into different <rpc-reply> messages
- D. All of the fields in one record MUST be put into one <rpc-reply> message.
- E. XML syntax MUST be complete in each fragmented message, so that each fragmentation could be parsed individually.
- F. If the record(s) of the child node(s)/table(s) and the parent node(s)/table(s) are replied in different fragmentations, the child node/table fragmentations MUST include the path and index information of all the ancestor node(s)/table(s) in a hierarchical mode.

- o Parameters

<discard/>: in <get-block> operation; if the <discard/> parameter is conveyed, it means the operation is terminated. Then it doesn't need to reply the remaining fragmentations.

#### o Successful Operation Reply

A <rpc-reply> message conveying a <data> element indicates the operation is successful.

If there exists a next fragment, then an set-id attribute MUST be included in the <rpc-reply> message. The attribute set-id is used to identify different fragment sets.

#### o Exception Handling

After the NETCONF server replies a fragment, if there is no corresponding Get-block request from the client in a reasonable period (the time valued to be specified in the future), then the server release the offset of the replying data and cannot use <get-block> operation anymore, and the remaining data needs to be replied.

Please refer to Appendix A.1 for an example.

### 4.2. Subtree Iteration

An "iterator" approach allows a list resource to be retrieved in chunks. An RPC function could be added to do the iteration operation.

Please refer to Appendix A.2 for an example.

There is a problem with rapidly changing lists (could get repeat entries on miss some entries).

### 4.3. Linked Replies

Another solution is to change or augment NETCONF at some point in time such that an <rpc> can lead to a sequence of <rpc-reply> with a suitable cancel mechanism. A simple approach is to add a linked-replies capability. If a server announces "linked-replies" capability and the client supports it as well, the client can add an additional parameter to an rpc to indicate the possible use of linked-replies.

Please refer to Appendix A.3 for an example.

This would address the concern of large data retrievals but would also allow long running asynchronous rpcs (the ping or traceroute example). This approach may lead to better support for asynchronous rpcs and rpcs that potentially return very large chunks of data than trying to solve this problem without enhancements of the rpc layer. Design details concerning data merging, error handling, how to send a cancel for a given link-id (e.g., by sending a new <rpc-cancel> message with a matching link-id) and whether it is necessary to negotiate linked rpc-reply sizes or whether it is good enough for the server to decide freely as it likes etc. need further study.

## 5. Security Considerations

TBD.

## 6. IANA Considerations

This draft does not request any IANA action.

## 7. Acknowledgements

Gang Yan and Shouchuan Yang made significant contribution to form the draft. Valuable comments were received from Andy Bierman, Martin Bjorklund, Juergen Schoenwaelder, Chong Feng and some other people in Netconf working group.

This document was produced using the xml2rfc tool [RFC2629].  
(initiallly prepared using 2-Word-v2.0.template.dot. )

## 8. References

### 8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.
- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

### 8.2. Informative References

- [DOM] "http://en.wikipedia.org/wiki/Document\_Object\_Model", .

[DOM-PARSING]        "http://www.w3.org/TR/DOM-Parsing/", .

[LIBXML]        "http://xmlsoft.org/", .

[SAX]        "http://en.wikipedia.org/wiki/Simple\_API\_for\_XML", .

[SAX-PARSING]        "http://www.saxproject.org/apidoc/org/xml/sax/  
Parser.html", .

## Appendix A. Examples of the Candidate Solutions

### A.1. <get-next> (RPC Fragmentation) Example

Example 1: Get the next fragment

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter type="subtree">
      <top xmlns="http://example.com/schema/1.2/config">
        <users/>
      </top>
    </filter>
  </get-config>
</rpc>

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0"
  xmlns:hw="http://www.huawei.com/NETCONF/capability/base/1.0"
hw:set-id="101">
  <data>
    <top xmlns="http://example.com/schema/1.2/config">
      <users>
        <user>
          <name>root</name>
          <type>superuser</type>
          <full-name>Charlie Root</full-name>
          <company-info>
            <dept>1</dept>
            <id>1</id>
          </company-info>
        </user>
        <!-- additional <user> elements appear here... -->
      </users>
    </top>
  </data>
</rpc-reply>
```

```

        </top>
      </data>
    </rpc-reply>

    <rpc message-id="102"
      xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
      <get-block xmlns="http://www.huawei.com/NETCONF/capability/base/1.0"
set-id="1">
        </get-block>
      </rpc>
      <rpc-reply message-id="102"
        xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0"
        xmlns:hw="http://www.huawei.com/NETCONF/capability/base/1.0"
hw:set-id="101">
        <data>
          <top xmlns="http://example.com/schema/1.2/config">
            <users>
              <user>
                <name>admin</name>
                <type>commonuser</type>
                <full-name>Jim Green</full-name>
                <company-info>
                  <dept>9</dept>
                  <id>90</id>
                </company-info>
              </user>
              <!-- additional <user> elements appear here... -->
            </users>
          </top>
        </data>
      </rpc-reply>

```

Example 2: Abandon the remaining fragments

```

    <rpc message-id="103"
      xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
      <get-block xmlns="http://www.huawei.com/NETCONF/capability/base/1.0" set-id
="1">
        <discard/>
      </get-block>
    </rpc>

    <rpc-reply message-id="103"
      xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
      <ok/>
    </rpc-reply>

```

Example 3: Following is an example of the rule f in Section 4.1.2. The child eTable is in a different message with the parents aTable->bTable->cTable->dTable. Then the path and index information

of all the ancestors MUST included in the search data.

```
<aTable>
  <aEntity>
    <aIndex1>
  </aEntity>
  <bTable>
    <bEntity>
  <bIndex1>
</bEntity>
</dTable>
<eTable>
  <eEntity>
    <eIndex1>
    <ef2>
    <ef3>
  </eEntity>
</dTable>
</bTable>
</aTable>
```

#### A.2. <get-list> (Subtree Iteration) Example

```
rpc get-list {
  input {
    leaf target {
      type schema-instance-identifier;
      description "Identifies subtree to retrieve.";
    }
    leaf start {
      type uint32;
      default 0;
      description "Number of entries to skip before starting retrieval";
    }
    leaf max-entries {
      type uint32 { range "1..max"; }
      default 25;
      description "Maximum number of list entries to retrieve";
    }
  }
  output {
    anyxml data {
      description "Contains the requested data";
    }
  }
}

<rpc>
```



```
<get-list>
  <target>/if:interfaces/if:interface</target>
</get-list>
</rpc>

<rpc-reply>
  <data>
    <interfaces>
      <interface>    .... first entry </interface>
      ...
      <interface>    .... 25th entry </interface>
    </interfaces>
  </data>
</rpc-reply>

<rpc>
  <get-list>
    <target>/if:interfaces/if:interface</target>
    <start>25</start>
  </get-list>
</rpc>

<rpc-reply>
  <data>
    <interfaces>
      <interface>    .... 26th entry </interface>
      ...
      <interface>    .... 50th entry </interface>
    </interfaces>
  </data>
</rpc-reply>
```

### A.3. Linked-replies Example

Here is what a new client might do if it wants to use linked replies:

```
<rpc message-id="101" link-id="123" xmlns="...">  
</rpc>
```

The server can either simply send an rpc-reply or it starts sending linked replies, e.g.:

```
<rpc-reply message-id="101" next-message-id="102" link-id="123" xmlns="...">  
</rpc-reply>
```

```
<rpc-reply message-id="102" next-message-id="103" link-id="123" xmlns="...">  
</rpc-reply>
```

```
<rpc-reply message-id="103" link-id="123" xmlns="...">  
</rpc-reply>
```

#### Authors' Addresses

Bing Liu  
Huawei Technologies  
Q14, Huawei Campus, No.156 Beiqing Road  
Hai-Dian District, Beijing, 100095  
P.R. China

Email: leo.liubing@huawei.com

Guangying Zheng  
Huawei Technologies  
Huawei Nanjing R&D Center  
101 Software Avenue, Yuhua District, Nanjing, Jiangsu, 210012  
P.R. China

Email: zhengguangying@huawei.com

Network Working Group  
Internet Draft  
Intended status: Informational  
Expires: January 3, 2015

B. Liu, Ed.  
G. Yan  
Huawei Technologies  
July 2, 2014

Multi-Instances Configuration Issue in NETCONF  
draft-liu-netconf-multi-instances-00.txt

## Abstract

This document puts together the NETCONF issues of configuring multiple instances including configuring multiple network element instances and multiple service instances. The main problem is how to configure the multiple instances in one NETCONF channel.

## 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 January 3, 2015.

## Copyright Notice

Copyright (c) 2011 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
2. Requirements Language and Terminology .....	3
3. Multiple Instance Management Requirements and Gaps .....	4
3.1. Multiple Instance Introduction .....	4
3.1.1. Multiple Network Element Instances (MNEI).....	4
3.1.2. Multiple Service Instances (MSI) .....	4
3.2. Management Requirements .....	5
3.2.1. MNEI Management Requirements .....	5
3.2.2. MSI Management Requirements .....	5
3.3. Gap Analysis .....	6
4. Solutions Discussion.....	7
4.1. NETCONF Context Extension .....	7
4.2. Common Service Container .....	8
4.3. Allowing Reusing One module in another .....	8
4.4. Common MSI Data Model .....	8
5. Security Considerations .....	8
6. IANA Considerations .....	8
7. References .....	9
7.1. Normative References .....	9
7.2. Informative References .....	9
8. Acknowledgments .....	9
Authors' Addresses .....	10

## 1. Introduction

In modern networks, according to the wide spread of network multiplex technologies such as VPN, a key network element (e.g. a PE router) or a basic service (e.g. routing) is usually separated as multiple instances to support network multiplex more efficiently.

NETCONF is been adopted by more and more network management systems to be the southbound interface of configuring the devices. When using NETCONF to configure the above mentioned key network element or service, sometimes it is actually to configure each instance rather than configure the whole network element. However, current NETCONF protocol and YANG models seem to not specifically consider the multiple instance configuration issues.

This document discusses the multiple instances configuring issue among the various scenarios. Multiple instances are separated as multiple network element instances and multiple service instances to be discussed respectively.

## 2. Requirements Language and Terminology

The key words "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 [RFC2119] when they appear in ALL CAPS. When these words are not in ALL CAPS (such as "should" or "Should"), they have their usual English meanings, and are not to be interpreted as [RFC2119] key words.

Terminology:

NEI: network element instance

SI: service instance

MNEI: multiple network element instance

MSI: multiple service instance

### 3. Multiple Instance Management Requirements and Gaps

#### 3.1. Multiple Instance Introduction

##### 3.1.1. Multiple Network Element Instances (MNEI)

###### o Logic System LS

A network element can be divided into multiple logical systems that each system can deal with a task independent from others. For example, a physical router can be divided into multiple logical systems (an LS on router could be also called LR: Logical Router) and each LS can run independent routing/MPLS protocols such as OSPF, IS-IS, BGP, RIP, LDP, and RSVP-TE .etc.

###### o Virtual System (VS)

A network element can also be divided into multiple virtual systems. The VSes also can do different tasks respectively. Different with LS, VS is basically virtualized in software level. If LS and VS are co-existing in one platform, one VS normally belongs to a specific LS and one LS could have multiple VSes.

For example, a physical router can also be divided into multiple virtual systems (then it could be called VR: virtual router). A VR can also run independent routing/MPLS protocols as OSPF, IS-IS, BGP, RIP, LDP, and RSVP-TE .etc. So in the perspective of processing a service, VS could be seen as equivalent to LS.

##### 3.1.2. Multiple Service Instances (MSI)

Following are several representative examples of multiple service instances.

###### o VRF Virtual Routing and Forwarding

VRF allows a router to have multiple independent routing tables so that they could deal with different routing tasks.

VRF could be seen as a container SI, which means it might contain other SIs such as following IGP/MTR SI.

###### o IGP Multi-Processes

OSPF and IS-IS can also have multiple instances in one device in the form of multiple processes of the same protocol stack. Each process

has its own independent routing area, path calculation and data stores like routing tables .etc.

- o MTR Multi-Topology Routing

From the perspective of services (e.g. voice, video, data traffic .etc), one physical network could be seen as different higher-layer topologies. MTR technology allows the forwarding capabilities to be separated for each topology. Specifically, MTR supports a separate multicast topology and multiple unicast topologies.

Normally, an MTR instance belongs to a specific VRF, and one VRF could contain multiple MTR instances.

### 3.2. Management Requirements

#### 3.2.1. MNEI Management Requirements

- o Req-1: Independently Managing Each MNEI

Management of multiple network element instances normally has the following two modes.

- Independent Management

- In independent management mode, the NMS considers each instance as a standalone network element. All the features of them, except board-level hardware features, could be operated independently from each other by the user.

- Central Management

- In central management mode, normally the NMS does not need to independently manage each instance in network element management level. However, there are two exceptions:

- 1) When the instance is at initialing or failure stage, the NMS still needs to distinguish and configure each instance.
    - 2) Although NMS doesn't need to independently manage each NEI, it still needs to independently manage different services. So when the services are bear on different NEIs, the NMS still needs to distinguish the NEIs in terms of distinguishing services.

In summary, both in independent management mode and central management mode, the NMS needs to independently manage each NEI.

#### 3.2.2. MSI Management Requirements

- o Req-2: Independently Managing Each MSI

Most of the time, the NMS also needs to distinguish different service instances, because they do different tasks which the NMS needs to configure respectively. So in NMS perspective, the requirement of independently managing each instance is similar between MSI and MNEI.

o Req-3: Crossing Configuration between Different SIs

Besides independently managing each SI for a standalone service, MSI has more complicity than MNEI that one SI might logically belong to another container SI. For example, an IGP instance might belong to a VRF instance. This requirement could have two choices as the following.

- Req-3a: Configure one SI under another container SI

As mentioned above, since SI-A might logically belong to SI-B, the straightforward way for the NMS to configure SI-A is to configure it under SI-B. For the above example, the NMS can configure an IGP instance under the hosting VRF configuration and does not need to particularly configure it in IGP.

- Req-3b: Configuring one SI binding to another container SI

Instead of Req-3a, Req-3b requires the NMS to directly configure the targeted SI and bind it to another container SI. For example, the NMS configures the IGP instance and clearly specify the IGP instance is for a specific VRF instance.

### 3.3. Gap Analysis

o Gap-1: NETCONF cannot configure multiple agents within one session

For R1 (independently manage each MNEI), in independent management mode, each instance would be assigned an IP address that the NMS could initial one NETCONF session for each thus there is no problem. However, when the instances are at initialing or failure stage, the NMS just couldn't connect to them directly to set up NETCONF sessions. The NMS will have to set up a NETCONF session with the main agent and configure different instances within this NETCONF session.

In central management mode, since the instances belong to one main network element, they often need to share the same IP address for communication to the NMS. According to R1, when the NMS needs to independently configure the instances, it also needs to configure them within the NETCONF session which is set between the NMS and the main device.



It is obvious that current NETCONF doesn't support configuring multiple agents within one NETCONF session, because NETCONF session is set up on a transport-layer (SSH, TLS .etc) channel, thus one NETCONF session can only correspond to one IP address.

o Gap-2: Lack of Multiple Instances Extension Capability in Data Models

For Req-2 (independently manage each MSI), since MSIs also need to share the same IP address for communication to the NMS, the NMS also needs to configure multiple services within one NETCONF session, which is similar with Gap-1. However, the bigger issue is that if a service model was not constructed as multiple instances at first, it is hard to be extended to support multiple instances. If the service models would support multiple instances internally, they will have to be re-constructed, which is a heavy burden for both standard revision and implementation. And the model re-construction needs to be one by one, thus it is not scalable.

o Gap-3a: Cannot Reuse one Module in another

For Req-3a (Configure one SI under another container SI), if the targeted service module could be directly reused when defining the container module, it would be very convenient to support the Req-3a operation. However, current YANG model doesn't have the capability.

o Gap-3b: Modules are not aware of MSI of the container module

For Req-3b (Configuring one SI binding to another container SI), it requires the targeted service module to be aware of the MSI of the container module so that the NMS could bind it to a specific container SI through a key.

#### 4. Solutions Discussion

##### 4.1. NETCONF Context Extension

As defined in [RFC3411], an SNMP context is a collection of management information accessible by an SNMP entity. An item of management information may exist in more than one context. An SNMP entity potentially has access to many contexts.

In NETCONF, similar context extension could be done that the multiple instances could be distinguished in one NETCONF session. To extend context, several key problems need to be solved as the following:

- To add a Context field in current NETCONF <rpc-request> message.
- There needs a kind of ID to identify each instance.
- The NMS needs to collect the list of the instance IDs so that it can specify which instance to be configured in the context field.

The context extension is mostly suitable for Gap-1 (NETCONF cannot configure multiple objects within one session). Especially in the MNEI cases, the context is just like simply pointing to another network element and nothing is relevant to data modules. The context extension can also be used for MSIs; however, the service module itself has to support multiple instances as the prediction.

#### 4.2. Common Service Container

Make a service container which indexes all the services in the network element into a single dedicated list. Thus, if some a service model needs to be extended to support multi-instance, it only needs the service container to reconstruct without reconstructing the data models.

This approach mainly fits the Gap-2. Gap-3b could also consider this approach.

#### 4.3. Allowing Reusing One module in another

As described in Req-3a, YANG could be extended to supporting module-level reuse rather than current group-level reuse.

#### 4.4. Common MSI Data Model

This issue was discussed in IETF NETCONF and NETMOD mailing list and the result of the discussion was to write a module defining the various types of service instances (e.g. VRF, IGP Multi, MTR .etc) and specifying the semantics of this kind of multi-instance. Then, if a data model needs to be aware of multiple service instance, multi-service-id (e.g. VRF-id) leafs will be added to the existing module via augments.

This approach basically fit the Gap-3b.

#### 5. Security Considerations

TBD.

#### 6. IANA Considerations

None.

## 7. References

### 7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.
- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

### 7.2. Informative References

- [RFC3411] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002.

## 8. Acknowledgments

Many valuable comments were received from Guangying Zheng and Xiaofeng Ji to improve the draft.

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Bing Liu  
Q14-4-A Building  
Huawei Technologies Co., Ltd  
Zhong-Guan-Cun Environment Protection Park, No.156 Beiqing Rd.  
Hai-Dian District, Beijing  
P.R. China

Email: leo.liubing@huawei.com

Gang Yan  
Q15-2-A Building  
Huawei Technologies Co., Ltd  
Zhong-Guan-Cun Environment Protection Park, No.156 Beiqing Rd.  
Hai-Dian District, Beijing  
P.R. China

Email: yangang@huawei.com

