Abstract

This document provides a problem statement, derives an initial gap analysis and illustrates a first set of solution approaches in regard to augmenting YANG data stores based on the CoAP Management Interface with YANG Push capabilities. A binary transfer mechanism for YANG Subscribed Notifications addresses both the requirements of constrained-node networks and the need for semantic interoperability via self-descriptiveness of the corresponding data in motion.

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 April 21, 2018.

Copyright Notice

Copyright (c) 2017 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
1. Context of the Problem

A binary transfer capability for YANG Subscribed Notifications [I-D.ietf-netconf-subscribed-notifications] based on YANG Push [I-D.ietf-netconf-yang-push] can be realized by using existing RFC and I-D work as building blocks. This section is intended to provide a corresponding overview of the existing ecosystem in order to identify gaps and therefore provide a problem statement.
1.1. Binary YANG transfer protocol

The CoAP Management Interface I-D (CoMI [I-D.ietf-core-comi]) defines operations for a YANG data store based on the Constrained Application Protocol (CoAP [RFC7252]). CoAP uses a request/response interaction model that is based on HTTP (similar to RESTCONF [RFC8040]) and allows for multiple transports, including UDP or TCP (see [I-D.ietf-core-coap-tcp-tls]). The Concise Binary Object Representation (CBOR [RFC7049]) is used for the serialization of data in motion in respect to CoAP operations and the data modeled with YANG [I-D.ietf-core-yang-cbor].

1.2. Device-Type Scope

[I-D.ietf-core-comi] states that CoAP "is designed for Machine to Machine (M2M) applications such as smart energy, smart city and building control. Constrained devices need to be managed in an automatic fashion to handle the large quantities of devices that are expected in future installations. Messages between devices need to be as small and infrequent as possible. The implementation complexity and runtime resources need to be as small as possible."

In addition, [I-D.ietf-core-comi] highlights that "CoMI and RESTCONF are intended to work in a stateless client-server fashion. They use a single round-trip to complete a single editing transaction, where NETCONF needs up to 10 round trips. To promote small messages, CoMI uses a YANG to CBOR mapping [I-D.ietf-core-yang-cbor] and numeric identifiers [I-D.ietf-core-sid] to minimize CBOR payloads and URI length."

In essence, via CoMI, a small sensor can emit a set of measurements as binary encoded YANG notifications, which would only add a minimal overhead to the data in motion, but would increase interoperability significantly due to the powerful and widely used semantics enabled by YANG (in contrast to a set of raw values that always require additional context information and imperative guidance to be managed and post-processed appropriately).

1.3. Subscriptions via CoAP

The CoAP pub/sub I-D defines a CoAP Subscribe operation [I-D.ietf-core-coap-pubsub] that is based on observing resources via the Observe option for the GET operation as defined in [RFC7641]. The CoAP pub/sub draft is intended to provide the capabilities and characteristics of MQTT via a CoAP based protocol. The only other CoAP operation that supports the Observe option is the FETCH operation defined in [RFC8132].
The Observe option creates a small corresponding state on the server side that eliminates the need for continuous polling of a resource via subsequent requests. Instead, subsequent responses including both the Observe option and using the token of the request that initiated the observation are returned when the observed resource changes. A subscription (i.e. the observe state retained on the server) can be discarded by the client by sending a correspond CoAP GET with Observe using an Observe parameter of 1 or simply by "forgetting" the observation and return a CoAP Reset after receiving a notification in the context of the subscription. A subscription can also be discarded by the server by sending a corresponding response that does not contain an Observe option.

The subscription used in CoAP pub/sub are used to subscribe to a topic provided by a CoAP broker REST API. YANG Push [I-D.ietf-netconf-yang-push] and corresponding YANG Subscribed Notifications are used to subscribe to data node updates provided by a YANG management interface. YANG subscriptions can include a filter expression (either a subtree expression or an XPATH expression). The encoding rules of XPATH expressions in CBOR are covered by [I-D.ietf-core-yang-cbor].

1.4. Configured Subscriptions and Call-Home

Configured subscriptions are basically static configuration that creates subscription state on the YANG data store when it is started and persists between boot-cycles without the need of a client to create that subscription state. In consequence, a configured subscription can result in unsolicited pushed notifications in respect to a YANG client.

A popular variant of the configured subscription as defined in [I-D.ietf-netconf-yang-push] is the Call Home procedure defined in [RFC8071]. In this approach, a Transport Layer application association with the YANG client is initiated by the YANG data store. After this "initial phase, in which the YANG server is acting like a client", the existing Transport Layer connection (or session, in case of, for example, TLS) is then used to the YANG client to initiate a subscription (i.e. the YANG client is initiating a dynamic subscription based on a pre-configured request retained and issued by the YANG data store).

1.5. Bootstrapping of Drop-Shipped Pledges

[I-D.ietf-anima-bootstrapping-keyinfra] highlights that effectively "to literally 'pull yourself up by the bootstraps' is an impossible action. Similarly, the secure establishment of a key infrastructure without external help is also an impossibility."
According to [I-D.ietf-anima-bootstrapping-keyinfra] the bootstrapping approach Call-Home has problems and limitations, which (amongst others) the draft itself is trying to address:

- the pledge requires realtime connectivity to the vendor service
- the domain identity is exposed to the vendor service (this is a privacy concern)
- the vendor is responsible for making the authorization decisions (this is a liability concern)

A Pledge in the context of [I-D.ietf-anima-bootstrapping-keyinfra] is "the prospective device, which has an identity installed by a third-party (e.g., vendor, manufacturer or integrator)."

A Pledge can be "drop-shipped", which refers to "the physical distribution of equipment containing the ‘factory default’ configuration to a final destination. In zero-touch scenarios there is no staging or pre-configuration during drop-ship."

In the scope of Call-Home as a part of YANG Push, either the factory default configuration of a drop-shipped Pledge that is a YANG data store would require to include the "home to Call Home" configuration or it has to be configured locally.

[I-D.ietf-netconf-zerotouch] is intended to provide more flexibility to the Call-Home procedure already - by allowing to stage connection attempts to a locally administered network and if that fails fall back to connecting to a remotely administered network. Alas, [I-D.ietf-netconf-zerotouch] is either prone to the same limitations as cited above or requires local configuration in order to find the home to Call-Home.

The "Join Registrar" defined by [I-D.ietf-anima-bootstrapping-keyinfra] mitigates the cited problems and limitation by introducing "a representative of the domain that is configured, perhaps autonomically, to decide whether a new device is allowed to join the domain. The administrator of the domain interfaces with a Join Registrar (and Coordinator) to control this process. Typically a Join Registrar is "inside" its domain."

2. Summary of the Problem Statement

Currently, the following gaps are identified:

- no CoAP Subscribe procedure for dynamic YANG subscriptions is standardized that is able to convey a filter expression and
potentially other metadata required in the context of a YANG Subscribed Notifications application association. Analogously, new payload types (e.g. a FETCH payload media-type) have to be defined.

- no CoAP Call Home feature is standardized to support a popular variant of configured YANG subscriptions.
- no general Call Home mechanism is standardized that enables the discovery of "a home to Call Home" or that would be able to deal with "changing homes" in a dynamic but secure manner.

In addition to the identified gaps, the semantics of metadata — if there are any — that have to be conveyed to or from a YANG data store in order to subscribe to a (filtered) YANG module or data node are not identified.

The problem statement could be summarized as follows:

"There is no complete solution based on CoAP to enable a freshly unpacked YANG data store ("drop-shipped pledge", e.g. the cliche light bulb) to discover an appropriate home it can than Call-Home to in a secure and trusted manner in order to push (un-)solicited subscribed notifications."

3. Potential Approaches and Solutions

There are multiple approaches that could lead to viable solutions that address the identified gaps. The following sections illustrate the general solution context and some of the most promising approaches.

3.1. YANG subscription variants

A YANG Push update subscription service both provides support for dynamic subscription (i.e. subscription state created by a client request, allowing for solicited push notifications in the context of an up-time cycle of the server) and configured subscription (i.e. subscription configuration retained on the server, allowing for unsolicited push notifications across up-time cycles of the server).

3.2. YANG Push via CoAP

The two CoAP operations that enable a subscription mechanism are GET and FETCH (i.e. by supporting the Observe option). Both operations are viable candidates for creating a CoAP-based YANG Push mechanism for CoMI.
3.3. Dynamic Subscriptions

Using CoAP, the client issuing the initial subscription request creates the subscription state. Examples are the GET or FETCH operation including an Observe option using an Observe parameter of 0 (zero).

3.3.1. YANG Push via GET

This usage scenario requires two consecutive operations. It is not possible to transfer a filter expression included in a GET operation. In consequence, a POST operation on a collection resource has to be conducted in order to convey a filter expression to the YANG data store, allowing it to return an URI that contains the data node information filtered in respect to the posted filter expression (encoded in CBOR).

This variant allows for multiple clients to observe a specific filtered data node without conducting a POST operation, if the corresponding URI is made known to other clients that did not conduct the POST operation or, for example, is canonically linked to/derivable from a filter expression.

3.3.2. YANG Push via FETCH

This usage scenario requires only one operation. A FETCH operation can include a body that is capable to contain a filter expression and potentially other metadata that might be required to establish a suitable subscription state on the YANG data store.

It might be possible that this variant could introduce a slight delay in respect to response time if providing a filtered resource requires a lot of computation time on a constrained device. I.e. the resource cannot be prepared "beforehand".

3.4. Configured Subscriptions

Using CoAP, the server retains configuration that creates subscription state when the YANG data store is started. The client has to have or gain knowledge of the CoAP tokens that are included in the responses created in the context of the subscription state create from server configuration.

3.4.1. Retaining the Content of a GET Operation as Configuration

This usage scenario "mimics" the receiving of a subscription request by storing the corresponding information that are relevant for creating a subscription state as configuration on the YANG data
store. I.e. the configuration would be including the YANG client IP address and the CoAP token to be used in the responses that convey the subscribed notifications.

This variant requires that the client also knows or gains knowledge of the corresponding CoAP token in order to not discard the incoming responses.

3.4.2. Call Home via CoAP

This usage scenario defines the Call Home procedure standardized in [RFC8071] as an additional capability of CoAP. DTLS or TLS state is initiated by the YANG data store and triggers a dynamic subscription procedure of the YANG client using the session initiated by the YANG data store.

3.4.3. Dynamic Home Discovery

This usage scenario is based on the Bootstrapping Remote Secure Key Infrastructures I-D [I-D.ietf-anima-bootstrapping-keyinfra] and EST over secure CoAP I-D [I-D.vanderstok-ace-coap-est] and requires the standardization of a general use of Join Registrars in the context of YANG data stores that support YANG Push via static subscriptions.

4. IANA considerations

This document includes no requests to IANA, but solutions drafts incubated via this document might.

5. Security Considerations

This document includes no security considerations, but solution drafts incubated via this document will.

6. Acknowledgements

Carsten Bormann, Klaus Hartke, Michel Veillette

7. Change Log

First version -00

8. Normative References

[I-D.ietf-anima-bootstrapping-keyinfra]

[I-D.ietf-core-coap-pubsub]

[I-D.ietf-core-coap-tcp-tls]

[I-D.ietf-core-comi]

[I-D.ietf-core-sid]

[I-D.ietf-core-yang-cbor]

[I-D.ietf-netconf-subscribed-notifications]

[I-D.ietf-netconf-yang-push]
[I-D.ietf-netconf-zerotouch]

[I-D.vanderstok-ace-coap-est]


Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt 64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

Birkholz, et al. Expires April 21, 2018
Tianran Zhou
Huawei
156 Beiqing Rd.
Beijing, Haidian District
China

Email: zhoutianran@huawei.com

Xufeng Liu
Jabil
8281 Greensboro Drive, Suite 200
McLean VA  22102
USA

Email: Xufeng_Liu@jabil.com

Eric Voit
Cisco Systems

Email: evoit@cisco.com
YANG PUSH Based Generalized Network Control Automation Problem Statement
draft-bryskin-netconf-automation-framework-00

Abstract

This document describes the objective of the YANG PUSH based
generalized network control automation framework.

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 19, 2018.

Copyright Notice

Copyright (c) 2017 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
1. Introduction

YANG "Custom Subscription to Event Notifications" model [I-D.ietf-netconf-subscribed-notifications] allows for a network client automation of network remote monitoring. Specifically, using this model, a network client can subscribe on and receive one or more data streams, each associated with one or more events defined by YANG model(s) governing the network’s YANG data store(s). The client can also tailor said streams to its needs by specifying filters on the streams contents, but, otherwise, the client has no control on the stream contents. For example, the client has no way of expanding a stream to carry additional information that was not defined to be a part of said stream.

YANG "Subscribing to YANG datastore push updates" model [I-D.ietf-netconf-yang-push], which is an augmentation of the "Custom Subscription to Event Notifications" model, defines a higher level of network remote monitoring automation – it allows for the client itself to define the origins, trigger/maintain conditions and contents of data streams to be sent by the network to the client. This capability is modeled via target-trigger-notify constructs, which allow for the client to specify data store nodes of interest and, possibly, sub-trees rooted by them (targets), conditions to trigger and maintain associated with them streams (e.g. particular change(s) in one or more of the nodes attributes), the contents of the streams and filters to further fine-tune the streams according to the client’s needs.

It could be observed that the notify part of the target-trigger-notify construct stands for "send me notification", which is one of, generally speaking, many actions the client might want the network to
perform, provided that the target-trigger condition holds. For example, instead of sending a notification with some pre-denied content, the client might want the network to perform:

a. immediate network re-configuration (e.g. modification of one or more attributes of one or more CONFIG=TRUE data store nodes);

b. scheduling one time or periodic such reconfigurations in the future;

c. calling an RPC defined by one of the YANG models supported by the network (e.g. calling network’s path computer to evaluate whether an alternative/more optimal path is available for a given connection);

d. Dynamic linking/unlinking parent and child data stores supported by the network;

e. etc.

It could also be observed that "periodic" and "on-change" are two of the conditions that the client might want. The conditions can be expanded to be a logical expression of other event states and some operational data states of the network., as well, as outputs of RPCs.

2. Objective

The main objective of the YANG PUSH Based Generalized Network Control Automation framework is to generalize the target-trigger-notify construct into event-condition-action construct, where:

**event**

a particular change in the network state explicitly defined by one of the YANG models supported by the network or implicitly defined by the client, which is constantly monitored by the network;

**condition**

a logical expression that is evaluated only once after the associated event is detected;

**action**

an operation (non-exhaustive list of which is described above) to be carried out by the network when the associated event is detected and the associated condition is met.

The client will be able to describe the desired network behavior by configuring with the network event-condition-action triplets as rules prior to any services provided by the network to the client. Such an
approach will take the client out of the network control loop, thus, changing the client’s role from being network’s "micro-manager" to being network’s "police officer", which interferes into network operations only in exceptional/unpredicted situations.

There are numerous benefits to such paradigm, including:

- lower latency, faster responsiveness of the network to various events/conditions;
- better scale (e.g. the client may control more networks because it does not have to monitor/micro-manage any of them);
- CPU and bandwidth savings due to the reduced amount of communication between the client and the network.

It is envisioned that the YANG PUSH Based Generalized Network Control Automation framework will fit well within "SUPA Policy-based Management Framework" [I-D.ietf-supaplolicy-based-management-framework], which will inherently provide a higher level of automation, for example, by:

a. combining multiple micro-conditions into a single macro-condition via a number of logical operations;

b. combining multiple micro-actions into a single transaction with a possibility of specifying policies with respect to handling errors/exceptions of each of the transaction components.

3. IANA Considerations

This document has no actions for IANA.

4. Security Considerations

This document does not define networking protocols and data, hence are not directly responsible for security risks.

5. Acknowledgements

6. References

6.1. Normative References


6.2. Informative References


Authors' Addresses

Igor Bryskin
Huawei Technologies
EMail: Igor.Bryskin@huawei.com

Xufeng Liu
Jabil
EMail: Xufeng_Liu@jabil.com
Abstract

This document defines a capability that allows to report discrepancies between management datastores in Netconf or Restconf servers that comply with the NMDA architecture. The capability is based on a set of RPCs that are defined as part of a YANG data model and that are intended to be used in conjunction with Netconf and Restconf.

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 May 3, 2018.

Copyright Notice

Copyright (c) 2017 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 revised Network Management Datastore Architecture (NMDA) [NMDA] introduces a set of new datastores that each hold YANG-defined data [RFC7950] and represent a different "viewpoint" on the data that is maintained by a server. New YANG datastores that are introduced include <intended>, which contains validated configuration data that a client application intends to be in effect, and <operational>, which contains at least conceptually operational state data (such as statistics) as well as configuration data that is actually in effect.

NMDA introduces in effect a concept of "lifecycle" for management data, allowing to clearly distinguish between data that is part of a configuration that was supplied by a user, configuration data that has actually been successfully applied and that is part of the operational state, and overall operational state that includes both applied configuration data as well as status and statistics.

As a result, data from the same management model can be reflected in multiple datastores. Clients need to specify the target datastore to be specific about which viewpoint of the data they want to access. This way, an application can differentiate whether they are (for example) interested in the configuration that has been applied and is actually in effect, or in the configuration that was supplied by a client and that is supposed to be in effect.

Due to the fact that data can propagate from one datastore to another, it is possibly for discrepancies to occur. Some of this is entirely expected, as there may be a time lag between when a configuration is given to the device and reflected in <intended>,

---

Clemm, et al.          Expires May 3, 2018
until when it actually takes effect and is reflected in <operational>. However, there may be cases when a configuration item that was to be applied may not actually take effect at all or needs an unusually long time to do so. This can be the case due to certain conditions not being met, resource dependencies not being resolved, or even implementation errors in corner conditions.

When configuration that is in effect is different from configuration that was applied, many issues can result. It becomes more difficult to operate the network properly due to limited visibility of actual status which makes it more difficult to analyze and understand what is going on in the network. Services may be negatively affected (for example, breaking a service instance resulting in service is not properly delivered to a customer) and network resources be misallocated.

Applications can potentially analyze any discrepancies between two datastores by retrieving the contents from both datastores and comparing them. However, in many cases this will be at the same time costly and extremely wasteful. It will also not be an effective approach to discover changes that are only "fleeting", or for that matter to distinguish between changes that are only fleeting from ones that are not and that may represent a real operational issue and inconsistency within the device.

This document introduces a YANG data model which defines RPCs, intended to be used in conjunction with NETCONF [RFC6241] or RESTCONF [RFC8040], that allow a client to request a server to compare two NMDA datastores and report any discrepancies. It also features a dampening option that allows to exclude discrepancies that are only fleeting from the report.

2. Key Words

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.

3. Definitions and Acronyms

NMDA: Network Management Datastore Architecture

RPC: Remote Procedure Call
4. Data Model Overview

At the core of the solution is a new management operation, <compare>, that allows to compare two datastores for the same data. The operation checks whether there are any discrepancies in values or in objects that are contained in either datastore, and returns any discrepancies as output. The output is returned in the format specified in YANG-Patch [RFC8072].

The YANG data model defines the <compare> operation as a new RPC. The operation takes the following input parameters:

- source: The source identifies the datastore that will serve as reference for the comparison, for example <intended>.
- target: The target identifies the datastore to compare against the source.
- filter-spec: This is a choice between different filter constructs to identify the portions of the datastore to be retrieved. It acts as a node selector that specifies which data nodes are within the scope of the comparison and which nodes are outside the scope. This allows a comparison operation to be applied only to a specific portion of the datastore that is of interest, such as a particular subtree. (The filter does not contain expressions that would match values data nodes, as this is not required by most use cases and would complicate the scheme, from implementation to dealing with race conditions.)
- dampening: Identifies the minimum time period for which a discrepancy must persist for it to be reported. The reporting of the output MAY correspondingly delayed by the dampening period. Implementations MAY thus run a comparison when the RPC is first invoked, then wait until after the dampening period to check whether any differences still persist. This parameter is conditional of a dampening being supported as a feature.

The operation provides the following output parameter:

- differences: This parameter contains the list of differences, encoded per RFC8072, i.e. specifying which patches would need to be applied to the source to produce the target.

As part of the differences, it will be useful to include "origin" metadata where applicable, specifically when the target datastore is <operational>. This can help explain the cause of a discrepancy, for example when a data item is part of <intended> but the origin in <operational> is reported as "system". How to best report "origin"
metadata is an item for further study, specifically whether it should be automatically returned per default or whether its reporting should be controlled using another RPC parameter.

The data model is defined in the ietf-nmda-compare YANG module. Its structure is shown in the following figure. The notation syntax follows [I-D.draft-ietf-netmod-yang-tree-diagrams].

module: ietf-nmda-compare

rpcs:
  +--x compare
      +--w input
          +--w source identityref
          +--w target identityref
          +--w (filter-spec)?
              |  +--w subtree-filter? <anydata>
              |  +--x subtree-filter?  <anydata>
              |  +--w xpath-filter?  yang:xpath1.0 {nc:xpath}? 
              |  +--w dampening?  yang:timeticks {cmp-dampening}? 
      +--ro output
          +--ro differences

Structure of ietf-nmda-compare

5. YANG Data Model

<CODE BEGINS> file "ietf-nmda-compare@2017-10-30.yang"

module ietf-nmda-compare {

  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-nmda-compare";

  prefix cp;

  import ietf-yang-types {
    prefix yang;
  }
  import ietf-datastores {
    prefix ds;
  }
  import ietf-yang-patch {
    prefix ypatch;
  }
  import ietf-netconf {
    prefix nc;

  }

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

Author: Alexander Clemm
  <mailto:ludwig@clemm.org>

Author: Yingzhen Qu
  <mailto:yingzhen.qu@huawei.com>

Author: Jeff Tantsura
  <mailto:jefftant.ietf@gmail.com>

description
  "The YANG data model defines a new operation, <compare>, that can be used to compare NMDA datastores."

revision 2017-10-30 {
  description
    "Initial revision";
  reference
    "RFC XXXX: Discrepancy detection between NMDA datastores";
}

feature cmp-dampening {
  description
    "This feature indicates that the ability to only report differences that pertain for a certain amount of time, as indicated through a dampening period, is supported."
}

/* RPC */
rpc compare {
  description
    "NMDA compare operation.";
  input {
    leaf source {
      type identityref {
        base ds:datastore;
      }
      mandatory true;
      description
        "The source datastore to be compared.";
    }
    leaf target {

type identityref {
    base ds:datastore;
}
mandatory true;
description
    "The target datastore to be compared.";
}
choice filter-spec {
    description
    "Identifies the portions of the datastores to be compared.";
}
    
anydata subtree-filter {
    description
    "This parameter identifies the portions of the target datastore to retrieve.";
    reference "RFC 6241, Section 6.";
}
leaf xpath-filter {
    if-feature nc:xpath;
    type yang:xpath1.0;
    description
    "This parameter contains an XPath expression identifying the portions of the target datastore to retrieve.";
}
}
leaf dampening {
    if-feature cmp-dampening;
    type yang:timeticks;
    default "0";
    description
    "The dampening period, in hundredths of a second, for the reporting of differences. Only differences that pertain for at least the dampening time are reported. Reporting of differences may be deferred by the dampening time. A value of 0 or omission of the leaf indicates no dampening.";
}
}
output {
    container differences {
    uses ypatch:yang-patch;
    description
    "The list of differences, encoded per RFC8072.";
    }
    }
}
6. IANA Considerations

6.1. Updates to the IETF XML Registry

This document registers one URI in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registration is requested:


Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

6.2. Updates to the YANG Module Names Registry

This document registers a YANG module in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the following registration is requested:

name: ietf-nmda-compare


prefix: cp

reference: RFC XXXX

7. Security Considerations

Comparing discrepancies between datastores requires a certain amount of processing resources at the server. An attacker could attempt to attack a server by making a high volume of discrepancy detection requests. Server implementations can guard against such scenarios in several ways. For one, they can implement NACM in order to require proper authorization for requests to be made. Second, server implementations can limit the number of requests that they serve in any one time interval, potentially rejecting requests made at a higher frequency than the implementation can reasonably sustain.

8. Acknowledgments

We thank Rob Wilton for valuable feedback and suggestions on an earlier revision of this document.
9. Normative References

[I-D.draft-ietf-netmod-yang-tree-diagrams]

[NMDA] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K.,
and R. Wilton, "Network Management Datastore
Architecture", October 2017,

[notif-sub]
Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard,
E., and A. Tripathy, "Custom subscription to event
notifications", October 2017,

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688,
DOI 10.17487/RFC3688, January 2004,

for the Network Configuration Protocol (NETCONF)", RFC 6020,
DOI 10.17487/RFC6020, October 2010,

and A. Bierman, Ed., "Network Configuration Protocol
(NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011,

RFC 7950, DOI 10.17487/RFC7950, August 2016,

Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017,


Authors' Addresses

Alexander Clemm
Futurewei Technologies, Inc.
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: ludwig@clemm.org

Yingzhen Qu
Futurewei Technologies, Inc.
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: yingzhen.qu@huawei.com

Jeff Tantsura
Futurewei Technologies, Inc.
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: jefftant.ietf@gmail.com
Abstract

This document defines a problem statement for Smart Filters for Push Updates. Smart Filters for Push Updates (referred to simply as "Smart Filters" in the context of this document) allows to filter push updates based on values of pushed objects and/or state, such as previous updates. Smart Filters provide an important building block for service assurance and network automation.

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 April 21, 2018.
1. Introduction

YANG-Push [yang-push] allows client applications to subscribe to continuous datastore updates without needing to poll. YANG-Push subscriptions allow client applications to select which datanodes are of interest. For this purpose, filters that act as node selectors are offered. However, what is currently not supported are filters that filter updates based on values, such as sending updates only when the value falls within a certain range. Also not supported are filters that would require additional state, such as sending updates only when the value exceeds a certain threshold for the first time but not again until the threshold is cleared. We refer to such filters as "smart filters", with further subcategories of "smart stateless filters" and "smart stateful filters", respectively.

Smart filters involve more complex subscription and implementation semantics than the simple selection filters that are currently offered as part of YANG-Push. They involve post processing of updates that goes beyond basic update generation for polling avoidance and place additional intelligence at the server. Because of this, smart filter functionality was not included in the YANG-Push specification, although it was recognized that YANG-Push could be
extended to include such functionality if needed. This is the purpose of this specification.

Smart filters facilitate service assurance, because they allow client applications to focus on "outliers" and updates that signify exceptions and conditions of interest have the biggest operational significance. They save network resources by avoiding the need to stream updates that would be discarded anyway, and allow applications to scale better since larger networks imply a larger amount of smart filtering operations delegated away from the application to the network. Smart filters also facilitate network automation as they constitute an important ingredient to specify triggers for automated actions.

2. Key Words

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.

3. Definitions and Acronyms

Smart Filter: A filter that involves some processing, such as comparing values or differentiating behavior depending on state.

TCA: Threshold Crossing Alert.

YANG-Push: A server capability that allows client applications to subscribe to network management datastore updates.

4. Problem Statement

YANG-Push provides client applications with the ability to subscribe to continuous updates from network management datastores, obviating the need to perform polling and resulting in more robust and efficient applications. However, many applications do not require every update, only updates that are of certain interest.

For example, an update concerning interface utilization may be only needed when a certain utilization level is breached. Sending continuous updates when utilization is low might divert processing resources away from updates regarding interfaces whose utilization level may reach a critical point that requires attention. Doing so will require a filter based on an object value. Even sending continuous updates when utilization is high may be too much and counterproductive. It may be sufficient to send an update when a
threshold is breached to raise a flag of attention, but then not to continue sending updates while the condition still persists but simply let the client application know when the threshold is cleared. This behavior cannot be accomplished simply by a value-based filter, but requires additional state to be maintained (so that the server has a memory whether or not the condition of a breached threshold has already been reported in prior update cycles).

What is needed are "Smart Filters" that provide the ability to apply filters based on object values, possibly also state. Smart Filters are useful for Service Assurance applications that need to monitor operational data for values that fall outside normal operational ranges. They are also useful for network automation, in which automated actions are automatically triggered based on when certain events in the network occur while certain conditions hold. A YANG-Push subscription with a smart filter can in effect act as a source for such events. Combined with an optional check for a condition when an event is observed, this can serve as the basis of action triggers.

Of course, it is possible to conceive filters that are very smart and powerful yet also very complex. While filters as defined in YANG-Push may be a tad too simple for the applications envisioned here, it is important to keep filters still simple enough to ensure broad implementation and support by networking devices. The smart filters defined in this effort intend to apply the "90/10" rule, aiming at the sweet spot that addresses 90% of use cases and deployment scenarios that can be addressed using 10% of the complexity. Where those filters are not sufficient, additional filters can be introduced outside this document.

It is proposed that Smart Filters for Push Updates will provide support for the following features:

- Support for smart filter extensions to YANG-Push subscriptions. The targeted model takes a "base" YANG-Push subscription and subjects updates to an additional filtering stage that is based on an object’s value.
  - Filters that match or compare the object value against a fixed term or expression.
  - Filters that match or compare an object value against the value of another object. (This feature is useful particularly in conjunction with possible extensions that allow to compute aggregates. Such an ability opens the possibility to compare an object’s current value to its mean, for example.)
Support for refined on-change update semantics that allow client to distinguish whether object values were omitted or included because the object was created or deleted, or because the object’s value fell outside filter range.

Support for selected stateful filters:

* This includes specifically support for generalized "threshold crossing alert" filters, or filters that provide an update only when an object’s value passes a filter for the first time, and not again until the object’s value passes a counter filter. In effect, the support involves attaching filter and counter filter to an object, including a switch at the object indicating which filter is in effect, and providing a distinction in the update which filter (e.g. onset of clear) was applied.

* It may include additional filters, such a "recent high water mark" filters that allow to specify a time horizon until the current high water mark clears. A recent high water mark filter sends an update to an object only if its new value is greater than the last value that had been previously reported.

In order to constrain complexity, it is proposed that the following items will be outside the scope, subject to discussion by the Working Group:

- Filters that involve freely programmable logic.
- Filters that aggregate or otherwise process information over time. An example would be filters that compute an aggregate over a time series of data, for example, an object’s average or top percentile value. (One way in which this can be accomplished is by defining a separate YANG module that allows to specify aggregates independent of any filtering, then asking users to subscribe to updates of the aggregate objects and applying filters there. The definition of such an aggregation module goes beyond the scope of the work defined here.)
- Filters that aggregate object’s values with those of other objects, such as the maximum or average from objects over a list, or that operate on a function of other objects. An example would be an object for interface utilization that gets computed from objects for interface speed and interface packet rate, with the packet rate object itself potentially computed from counter snapshots that are taken at different times. (One way in which this can be accomplished is by defining a separate YANG module that allows to define objects that compute such functions akin to
5. IANA Considerations
Not applicable

6. Security Considerations

The application of smart filters requires a certain amount of processing resources at the server. An attacker could attempt to attack a server by creating YANG-push subscriptions with a large number of complex smart filters in an attempt to diminish server resources. Server implementations can guard against such scenarios in several ways. For one, they can implement NACM [RFC6536] in order to require proper authorization for requests to be made. Second, server implementations can reject requests made for a larger number of smart filters than the implementation can reasonably sustain.

7. Normative References

[notif-sub]


Subscribing to YANG datastore push updates

Clemm, A., Voit, E., Gonzalez Prieto, A., Tripathy, A.,
Nilsen-Nygaard, E., Bierman, A., and B. Lengyel,
"Subscribing to YANG datastore push updates", August 2017,

Authors’ Addresses

Alexander Clemm
Futurewei Technologies, Inc.
2330 Central Expressway
Santa Clara, CA 95050
USA
Email: ludwig@clemm.org

Eric Voit
Cisco Systems
Email: evoit@cisco.com

Xufeng Liu
Jabil
Email: Xufeng_Liu@jabil.com

Igor Bryskin
Huawei
Email: Igor.Bryskin@huawei.com

Tianran Zhou
Huawei
Email: zhoutianran@huawei.com

Guangying Zheng
Huawei
Email: zhengguangying@huawei.com
Abstract

This document defines a YANG 1.1 module called "ietf-keystore" that enables centralized configuration of asymmetric keys and their associated certificates, and notification for when configured certificates are about to expire.

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

- "VVVV" --> the assigned 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:

- "2018-06-04" --> the publication date of this draft

The following Appendix section is to be removed prior to publication:

- Appendix A. 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 December 6, 2018.

Copyright Notice

Copyright (c) 2018 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 .......................... 3
2. Requirements Language .................. 3
3. The Keystore Model ..................... 4
   3.1. Tree Diagram ....................... 4
   3.2. Example Usage ...................... 6
   3.3. YANG Module ....................... 12
4. Security Considerations ................ 21
5. IANA Considerations ................... 23
   5.1. The IETF XML Registry .............. 23
   5.2. The YANG Module Names Registry .... 23
6. References ............................ 23
   6.1. Normative References .............. 23
   6.2. Informative References ............ 24
Appendix A. Change Log .................... 26
   A.1. 00 to 01 ............................ 26
   A.2. 01 to 02 ............................ 26
   A.3. 02 to 03 ............................ 26
   A.4. 03 to 04 ............................ 26
   A.5. 04 to 05 ............................ 27
Acknowledgements .......................... 27
Author’s Address .......................... 27
1. Introduction

This document defines a YANG 1.1 [RFC7950] module called "ietf-keystore" that enables centralized configuration of asymmetric keys and their associated certificates, and notification for when configured certificates are about to expire.

This module also defines Six groupings designed for maximum reuse. These groupings include one for the public half of an asymmetric key, one for both the public and private halves of an asymmetric key, one for both halves of an asymmetric key and a list of associated certificates, one for an asymmetric key that may be configured locally or via a reference to an asymmetric key in the keystore, one for a trust anchor certificate and, lastly, one for an end entity certificate.

Special consideration has been given for systems that have cryptographic hardware, such as a Trusted Protection Module (TPM). These systems are unique in that the cryptographic hardware completely hides the private keys and must perform all private key operations. To support such hardware, the "private-key" can be the special value "hardware-protected" and the actions "generate-private-key" and "generate-certificate-signing-request" can be used to direct these operations to the hardware.

This document is compliant with Network Management Datastore Architecture (NMDA) [RFC8342]. For instance, to support keys and associated certificates installed during manufacturing (e.g., for an IDevID [Std-802.1AR-2009] certificate), it is expected that such data may appear only in <operational>.

While only asymmetric keys are currently supported, the module has been designed to enable other key types to be introduced in the future.

The module does not support protecting the contents of the keystore (e.g., via encryption), though it could be extended to do so in the future.

It is not required that a system has an operating system level keystore utility to implement this module.

2. 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
3. The Keystore Model

3.1. Tree Diagram

This section provides a tree diagrams [RFC8340] for the "ietf-keystore" module that presents both the protocol-accessible "keystore" as well the all the groupings intended for external usage.

module: ietf-keystore
	+-rw keystore
	  +--rw asymmetric-keys
	      +--rw asymmetric-key* [name]
	          |  +--rw name string
	          |  +--rw algorithm
	          |  |       ct:key-algorithm-ref
	          |  +--rw public-key binary
	          |  +--rw private-key union
	          +--rw certificates
	              |  +--rw certificate* [name]
	              |      |  +--rw name string
	              |      |  +--rw cert
	              |      |     |       ct:end-entity-cert-cms
	              |      |     |     |       n certificate-expiration
	              |      |     |     |       +-- expiration-date? yang:date-and-time
	              |      +-- x generate-certificate-signing-request
	              |      |  +-- w input
	              |      |  |  +-- w subject binary
	              |      |  |  +-- w attributes? binary
	              |      +-- ro output
	              |          +-- ro certificate-signing-request binary
	              +-- x generate-asymmetric-key
	                  +-- w input
	                      |  +-- w name string
	                      +-- w algorithm ct:key-algorithm-ref

grouping end-entity-cert-grouping
	+- cert ct:end-entity-cert-cms
	 +-- n certificate-expiration
	     +-- expiration-date? yang:date-and-time

grouping local-or-keystore-end-entity-certificate-grouping
	+- (local-or-keystore)
	 |  +-- (local)
	 |      |  +-- algorithm ct:key-algorithm-ref
	 |      |  +-- public-key binary
	 |      +-- private-key union

Watsen                  Expires December 6, 2018                [Page 4]
```xml
<keystore>
  <cert>
    <ct:end-entity-cert-cms/>
  </cert>
  <n certificate-expiration>
    <expiration-date? yang:date-and-time/>
  </n>
  <keystore-implemented/>
  <reference/>
</keystore>
```

```xml
<local-or-keystore>
  <algorithm>
    <ct:key-algorithm-ref binary/>
  </algorithm>
  <private-key>
    <union/>
  </private-key>
  <certificates>
    <certificate* [name]>
      <name?> string/>
      <cert ct:end-entity-cert-cms/>
    </certificate*>
    <n certificate-expiration>
      <expiration-date? yang:date-and-time/>
    </n>
    <generate-certificate-signing-request>
      <input>
        <subject binary/>
        <attributes? binary/>
      </input>
      <ro output>
        <certificate-signing-request binary/>
      </ro>
    </generate-certificate-signing-request>
  </certificates>
  <keystore-implemented/>
  <reference/>
</local-or-keystore>
```

```xml
<trust-anchor-cert-grouping>
  <cert ct:trust-anchor-cert-cms/>
</trust-anchor-cert-grouping>
```

```xml
<asymmetric-key-pair-grouping>
  <algorithm ct:key-algorithm-ref/>
  <public-key binary/>
  <private-key union/>
</asymmetric-key-pair-grouping>
```

```xml
<asymmetric-key-pair-with-certs-grouping>
  <algorithm ct:key-algorithm-ref/>
  <public-key binary/>
  <private-key union/>
  <certificates>
    <certificate* [name]>
      <name?> string/>
      <cert ct:end-entity-cert-cms/>
    </certificate*>
    <n certificate-expiration>
      <expiration-date? yang:date-and-time/>
    </n>
    <generate-certificate-signing-request/>
  </certificates>
</asymmetric-key-pair-with-certs-grouping>
```
3.2. Example Usage

The following example illustrates what a fully configured keystore might look like in <operational>, as described by Section 5.3 in [RFC8342]. This datastore view illustrates data set by the manufacturing process alongside conventional configuration. This keystore instance has three keys, two having one associated certificate and one having two associated certificates.

```xml
<keystore xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore"
         xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
  <asymmetric-keys>
    <asymmetric-key or:origin="or:intended">
      <name>ex-rsa-key</name>
      <algorithm>ct:rsa1024</algorithm>
      <private-key>base64encodedvalue==</private-key>
      <public-key>base64encodedvalue==</public-key>
      <certificates>
        <certificate>
          <name>ex-rsa-cert</name>
          <cert>base64encodedvalue==</cert>
        </certificate>
      </certificates>
    </asymmetric-key>
    <asymmetric-key or:origin="or:intended">
      <name>tls-ec-key</name>
      <algorithm>ct:secp256r1</algorithm>
      <private-key>base64encodedvalue==</private-key>
      <public-key>base64encodedvalue==</public-key>
      <certificates>
        <certificate>
          <name>tls-ec-cert</name>
          <cert>base64encodedvalue==</cert>
        </certificate>
      </certificates>
    </asymmetric-key>
  </asymmetric-keys>
</keystore>
```
<asymmetric-key or:origin="or:system">
  <name>tpm-protected-key</name>
  <algorithm>ct:rsa2048</algorithm>
  <private-key>hardware-protected</private-key>
  <public-key>base64encodedvalue==</public-key>
  <certificates>
    <certificate>
      <name>builtin-idevid-cert</name>
      <cert>base64encodedvalue==</cert>
    </certificate>
    <certificate or:origin="or:intended">
      <name>my-ldevid-cert</name>
      <cert>base64encodedvalue==</cert>
    </certificate>
  </certificates>
</asymmetric-key>
</asymmetric-keys>
</keystore>

The following example illustrates the "generate-private-key" action in use with the NETCONF protocol.
The following example illustrates the "generate-certificate-signing-request" action in use with the NETCONF protocol.
The following example illustrates the "certificate-expiration" notification in use with the NETCONF protocol.
The following example module has been constructed to illustrate the "local-or-keystore-asymmetric-key-grouping" grouping defined in the "ietf-keystore" module.
module ex-keystore-usage {
  yang-version 1.1;

  namespace "http://example.com/ns/example-keystore-usage";
  prefix "eku";

  import ietf-keystore {
    prefix ks;
    reference
      "RFC VVVV: YANG Data Model for a 'Keystore' Mechanism";
  }

  organization
    "Example Corporation";

  contact
    "Author: YANG Designer <mailto:yang.designer@example.com>";

  description
    "This module illustrates the grouping defined in the keystore
draft called 'local-or-keystore-asymmetric-key-grouping'.";

  revision "YYYY-MM-DD" {
    description
      "Initial version";
    reference
      "RFC XXXX: YANG Data Model for a 'Keystore’ Mechanism";
  }

  container keys {
    description
      "A container of keys.";
    list key {
      key name;
      leaf name {
        type string;
        description
          "An arbitrary name for this key.";
      }
      uses ks:local-or-keystore-asymmetric-key-grouping;
      description
        "A key which may be configured locally or be a reference to
         a key in the keystore.";
    }
  }
}
The following example illustrates what two configured keys, one local and the other remote, might look like. This example consistent with other examples above (i.e., the referenced key is in an example above).

```xml
<keys xmlns="http://example.com/ns/example-keystore-usage">
  <key>
    <name>locally-defined key</name>
      ct:secp521r1
    </algorithm>
    <private-key;base64encodedvalue==</private-key>
    <public-key;base64encodedvalue==</public-key>
  </key>
  <key>
    <name>keystore-defined key</name>
    <reference>ex-rsa-key</reference>
  </key>
</keys>
```

3.3. YANG Module

This YANG module imports modules defined in [RFC6536], [RFC6991], and [I-D.ietf-netconf-crypto-types]. This module uses data types defined in [RFC2986], [RFC3447], [RFC5652], [RFC5915], [RFC6125], and [ITU.X690.2015].

```yang
module ietf-keystore {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-keystore";
  prefix "ks";

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-crypto-types {
    prefix ct;
    reference
      "RFC CCCC: Common YANG Data Types for Cryptography";
  }

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

contact
"WG Web: <http://datatracker.ietf.org/wg/netconf/>
WG List: <mailto:netconf@ietf.org>
Author: Kent Watsen
<mailto:kwatsen@juniper.net>");

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

Copyright (c) 2018 IETF Trust and the persons identified
as authors of the code. All rights reserved.

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

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

revision "2018-06-04" {
  description
  "Initial version";
  reference
  "RFC VVVV: YANG Data Model for a ‘Keystore’ Mechanism";
}

// Features

feature keystore-implemented {
  description
  "The ‘keystore-implemented’ feature indicates that the server
  implements the keystore, and therefore groupings defined in
  this module that reference the keystore are usable.";
}

// Typedefs
typedef asymmetric-key-ref {
  type leafref {

typedef asymmetric-key-certificate-ref {
  type leafref {
    path "/ks:keystore/ks:asymmetric-keys/ks:asymmetric-key"
    + "/ks:certificates/ks:certificate/ks:name";
    require-instance false;
  }
  description
  "This typedef enables modules to easily define a reference
  to a specific certificate associated with an asymmetric key
  stored in the keystore. The require instance attribute is false
  to enable the referencing of certificates that exist only in <operational>.";
  reference
  "RFC 8342: Network Management Datastore Architecture (NMDA)";
}

// Groupings
//
// These groupings are factored out more than needed for
// reusability purposes.

grouping public-key-grouping {
  description
  "A public key.";
  leaf algorithm {
    type ct:key-algorithm-ref;
    mandatory true;
    description
    "Identifies the key's algorithm. More specifically,
    this leaf specifies how the 'public-key' binary leaf
    is encoded.";
    reference
    "RFC CCCC: Common YANG Data Types for Cryptography";
  }
}
leaf public-key {
  type binary;
  mandatory true;
  description
  "A binary that contains the value of the public key. The interpretation of the content is defined by the key algorithm. For example, a DSA key is an integer, an RSA key is represented as RSAPublicKey as defined in RFC 3447, and an Elliptic Curve Cryptography (ECC) key is represented using the 'publicKey' described in RFC 5915.";
  reference
  "RFC 3447: Public-Key Cryptography Standards (PKCS) #1:
RFC 5915: Elliptic Curve Private Key Structure.";
}

grouping asymmetric-key-pair-grouping {
  description
  "A private/public key pair.";
  uses public-key-grouping;
  leaf private-key {
    type union {
      type binary;
      type enumeration {
        enum "hardware-protected" {
          description
          "The private key is inaccessible due to being protected by a cryptographic hardware module (e.g., a TPM).";
        }
      }
    }
    mandatory true;
    description
    "A binary that contains the value of the private key. The interpretation of the content is defined by the key algorithm. For example, a DSA key is an integer, an RSA key is represented as RSAPrivateKey as defined in RFC 3447, and an Elliptic Curve Cryptography (ECC) key is represented as ECPrivateKey as defined in RFC 5915.";
    reference
    "RFC 3447: Public-Key Cryptography Standards (PKCS) #1:
RFC 5915: Elliptic Curve Private Key Structure.";
  }
}
grouping trust-anchor-cert-grouping {
  description
  "A certificate, and a notification for when it might expire.";
  leaf cert {
    type ct:trust-anchor-cert-cms;
    mandatory true;
    description
    "The binary certificate data for this certificate.";
    reference
    "RFC YYYY: Common YANG Data Types for Cryptography";
  }
}

grouping end-entity-cert-grouping {
  description
  "A certificate, and a notification for when it might expire.";
  leaf cert {
    type ct:end-entity-cert-cms;
    mandatory true;
    description
    "The binary certificate data for this certificate.";
    reference
    "RFC YYYY: Common YANG Data Types for Cryptography";
  }
}

notification certificate-expiration {
  description
  "A notification indicating that the configured certificate
  is either about to expire or has already expired. When to
  send notifications is an implementation specific decision,
  but it is RECOMMENDED that a notification be sent once a
  month for 3 months, then once a week for four weeks, and
  then once a day thereafter until the issue is resolved.";
  leaf expiration-date {
    type yang:date-and-time;
    //mandatory true;
    description
    "Identifies the expiration date on the certificate.";
  }
}

grouping asymmetric-key-pair-with-certs-grouping {
  description
  "A private/public key pair and associated certificates.";
  uses asymmetric-key-pair-grouping;
  container certificates {
    description
    "Certificates associated with this asymmetric key.
  
Watsen Expires December 6, 2018
More than one certificate supports, for instance, a TPM-protected asymmetric key that has both IDevID and LDevID certificates associated.

```ini
list certificate {
    key name;
    description "A certificate for this asymmetric key.";
    leaf name {
        type string;
        description "An arbitrary name for the certificate.";
    }
    uses end-entity-cert-grouping;
} // end certificate
} // end certificates

action generate-certificate-signing-request {
    description "Generates a certificate signing request structure for the associated asymmetric key using the passed subject and attribute values. The specified assertions need to be appropriate for the certificate’s use. For example, an entity certificate for a TLS server SHOULD have values that enable clients to satisfy RFC 6125 processing.";
    input {
        leaf subject {
            type binary;
            mandatory true;
            description "The ‘subject’ field per the CertificationRequestInfo structure as specified by RFC 2986, Section 4.1 encoded using the ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690.";
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).";
        }
        leaf attributes {
            type binary;
            description "The ‘attributes’ field from the structure CertificationRequestInfo as specified by RFC 2986,
";
        }
    }
}
```
Section 4.1 encoded using the ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690.

reference
"RFC 2986:
PKCS #10: Certification Request Syntax
Specification Version 1.7.
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)."

output {
leaf certificate-signing-request {
  type binary;
  mandatory true;
  description
  "A CertificationRequest structure as specified by RFC 2986, Section 4.2 encoded using the ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690."
  reference
  "RFC 2986:
   PKCS #10: Certification Request Syntax
   Specification Version 1.7.
  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)."

}
} // end output
} // end generate-certificate-signing-request

grouping local-or-keystore-asymmetric-key-grouping {
  description
  "A grouping that expands to allow the key to be either stored locally within the using data model, or be a reference to an asymmetric key stored in the keystore."
  choice local-or-keystore {
    mandatory true;
    case local {
      uses asymmetric-key-pair-grouping;
    }
    case keystore {

Watsen Expires December 6, 2018 [Page 18]
if-feature "keystore-implemented";
leaf reference {
  type ks:asymmetric-key-ref;
  mandatory true;
  description
    "A reference to a value that exists in the keystore.";
}

description
  "A choice between an inlined definition and a definition
  that exists in the keystore.";
}

grouping local-or-keystore-asymmetric-key-with-certs-grouping {
  description
    "A grouping that expands to allow the key to be either stored
    locally within the using data model, or be a reference to an
    asymmetric key stored in the keystore.";
  choice local-or-keystore {
    mandatory true;
    case local {
      uses asymmetric-key-pair-with-certs-grouping;
    }
    case keystore {
      if-feature "keystore-implemented";
      leaf reference {
        type ks:asymmetric-key-ref;
        mandatory true;
        description
          "A reference to a value that exists in the keystore.";
      }
      description
        "A choice between an inlined definition and a definition
        that exists in the keystore.";
    }
  }
}

grouping local-or-keystore-end-entity-certificate-grouping {
  description
    "A grouping that expands to allow the end-entity certificate
    (and the associated private key) to be either stored locally
    within the using data model, or be a reference to a specific
    certificate in the keystore.";
  choice local-or-keystore {
    mandatory true;
    case local {

uses ks:asymmetric-key-pair-grouping;
uses ks:end-entity-cert-grouping;
}
case keystore {
  if-feature "keystore-implemented";
  leaf reference {
    type ks:asymmetric-key-certificate-ref;
    mandatory true;
    description
    "A reference to a value that exists in the keystore.";
  }
  description
  "A choice between an inlined definition and a definition
  that exists in the keystore.";
}

// protocol accessible nodes
container keystore {
  description
  "The keystore contains a list of keys.";

  container asymmetric-keys {
    description
    "A list of asymmetric keys.";
    list asymmetric-key {
      key name;
      description
      "An asymmetric key.";
      leaf name {
        type string;
        description
        "An arbitrary name for the asymmetric key.";
      }
      uses asymmetric-key-pair-with-certs-grouping;
    } // end asymmetric-key
  } // end asymmetric-keys

  action generate-asymmetric-key {
    description
    "Requests the device to generate an asymmetric key using
    the specified asymmetric key algorithm. This action is
    primarily to support cryptographic processors that must
    generate the asymmetric key themselves. The resulting
    asymmetric key is considered operational state and hence
    present only in <operational>.";
  }
} // end keystore
input {
  leaf name {
    type string;
    mandatory true;
    description
    "The name the asymmetric key should have when listed
    in /keystore/asymmetric-keys/asymmetric-key, in
    <operational>.";
  }
  leaf algorithm {
    type ct:key-algorithm-ref;
    mandatory true;
    description
    "The algorithm to be used when generating the
    asymmetric key.";
    reference
    "RFC CCCC: Common YANG Data Types for Cryptography";
  } // end generate-asymmetric-key
} // end asymmetric-keys
} // end keystore

4. Security Considerations

The YANG module defined in this document is designed to be accessed
via YANG based management protocols, such as NETCONF [RFC6241] and
RESTCONF [RFC8040]. Both of these protocols have mandatory-to-
implement secure transport layers (e.g., SSH, TLS) with mutual
authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means
to restrict access for particular users to a pre-configured subset of
all available protocol operations and content.

There are a number of data nodes defined in this YANG module that are
writable/creatable/deletable (i.e., config true, which is the
default). These data nodes may be considered sensitive or vulnerable
in some network environments. Write operations (e.g., edit-config)
to these data nodes without proper protection can have a negative
effect on network operations. These are the subtrees and data nodes
and their sensitivity/vulnerability:
The entire data tree defined by this module is sensitive to write operations. For instance, the addition or removal of keys, certificates, trusted anchors, etc., can dramatically alter the implemented security policy. However, no NACM annotations are applied as the data SHOULD be editable by users other than a designated 'recovery session'.

When writing this node, implementations MUST ensure that the strength of the key being configured is not greater than the strength of the underlying secure transport connection over which it is communicated. Implementations SHOULD fail the write-request if ever the strength of the private key is greater than the strength of the underlying transport, and alert the client that the strength of the key may have been compromised. Additionally, when deleting this node, implementations SHOULD automatically (without explicit request) zeroize these keys in the most secure manner available, so as to prevent the remnants of their persisted storage locations from being analyzed in any meaningful way.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

This node is additionally sensitive to read operations such that, in normal use cases, it should never be returned to a client. The best reason for returning this node is to support backup/restore type workflows. However, no NACM annotations are applied as the data SHOULD be editable by users other than a designated ‘recovery session’.

Some of the operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

For this action, it is RECOMMENDED that implementations assert channel binding (RFC5056), so as to ensure that the application layer that sent the request is the same as the device authenticated when the secure transport layer was established.

This document uses PKCS #10 [RFC2986] for the "generate-certificate-signing-request" action. The use of Certificate Request Message

Watsen
Expires December 6, 2018
Format (CRMF) [RFC4211] was considered, but it was unclear if there was market demand for it. If it is desired to support CRMF in the future, placing a "choice" statement in both the input and output statements, along with an "if-feature" statement on the CRMF option, would enable a backwards compatible solution.

5. IANA Considerations

5.1. The IETF XML Registry

This document registers one URI in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registration is requested:

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

5.2. The YANG Module Names Registry

This document registers one YANG module in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the following registration is requested:

name: ietf-keystore
prefix: ks
reference: RFC VVVV

6. References

6.1. Normative References

[I-D.ietf-netconf-crypto-types]
Watsen, K., "Common YANG Data Types for Cryptography",
draft-ietf-netconf-crypto-types-00 (work in progress),
June 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/>.
6.2. Informative References


6.2. Informative References


Appendix A. Change Log

A.1. 00 to 01
   o Replaced the ‘certificate-chain’ structures with PKCS#7 structures. (Issue #1)
   o Added ‘private-key’ as a configurable data node, and removed the ‘generate-private-key’ and ‘load-private-key’ actions. (Issue #2)
   o Moved ‘user-auth-credentials’ to the ietf-ssh-client module. (Issues #4 and #5)

A.2. 01 to 02
   o Added back ‘generate-private-key’ action.
   o Removed ‘RESTRICTED’ enum from the ‘private-key’ leaf type.
   o Fixed up a few description statements.

A.3. 02 to 03
   o Changed draft’s title.
   o Added missing references.
   o Collapsed sections and levels.
   o Added RFC 8174 to Requirements Language Section.
   o Renamed ‘trusted-certificates’ to ‘pinned-certificates’.
   o Changed ‘public-key’ from config false to config true.
   o Switched ‘host-key’ from OneAsymmetricKey to definition from RFC 4253.

A.4. 03 to 04
   o Added typedefs around leafrefs to common keystore paths
   o Now tree diagrams reference ietf-netmod-yang-tree-diagrams
   o Removed Design Considerations section
   o Moved key and certificate definitions from data tree to groupings
A.5. 04 to 05

- FIXME
- FIXME
- FIXME

Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): Andy Bierman, Martin Bjorklund, Benoit Claise, Mehmet Ersue, Balazs Kovacs, David Lamparter, Alan Luchuk, Ladislav Lhotka, Mahesh Jethanandani, Radek Krejci, Reshad Rahman, Tom Petch, Juergen Schoenwaelder, Phil Shafer, Sean Turner, Eric Voit, Bert Wijnen, and Liang Xia.

Author’s Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net
NETCONF Client and Server Models
draft-ietf-netconf-netconf-client-server-06

Abstract

This document defines two YANG modules, one module to configure a NETCONF client and the other module to configure a NETCONF server. Both modules support both the SSH and TLS transport protocols, and support both standard NETCONF and NETCONF Call Home connections.

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.

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

- I-D.ietf-netconf-keystore
- I-D.ietf-netconf-ssh-client-server
- I-D.ietf-netconf-tls-client-server

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

- "XXXX" --> the assigned RFC value for this draft
- "YYYY" --> the assigned RFC value for I-D.ietf-netconf-ssh-client-server
- "ZZZZ" --> the assigned RFC value for I-D.ietf-netconf-tls-client-server

Artwork in this document contains placeholder values for the date of publication of this draft. Please apply the following replacement:
The following Appendix section is to be removed prior to publication:

- Appendix A. Change Log
1. Introduction

This document defines two YANG [RFC7950] modules, one module to configure a NETCONF [RFC6241] client and the other module to configure a NETCONF server. Both modules support both NETCONF over SSH [RFC6242] and NETCONF over TLS [RFC7589] and NETCONF Call Home connections [RFC8071].

2. 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 BCP
3. The NETCONF Client Model

The NETCONF client model presented in this section supports both
clients initiating connections to servers, as well as clients
listening for connections from servers calling home.

This model supports both the SSH and TLS transport protocols, using
the SSH client and TLS client groupings defined in
[I-D.ietf-netconf-ssh-client-server] and
[I-D.ietf-netconf-tls-client-server] respectively.

All private keys and trusted certificates are held in the keystore
model defined in [I-D.ietf-netconf-keystore].

YANG feature statements are used to enable implementations to
advertise which parts of the model the NETCONF client supports.

3.1. Tree Diagram

The following tree diagram [RFC8340] provides an overview of the data
model for the "ietf-netconf-client" module. Just the container is
displayed below, but there is also a reusable grouping by the same
name that the container is using.

```
module: ietf-netconf-client
   +--rw netconf-client
      +--rw initiate! {initiate}?
         +--rw netconf-server* [name]
            +--rw name                  string
            +--rw endpoints
               +--rw endpoint* [name]
                  +--rw name                  string
                  +--rw (transport)
                     +--:(ssh) {ssh-initiate}?
                        +--rw ssh
                           +--rw address?            inet:host
                           +--rw port?               inet:port-number
                           +--rw client-identity
                              +--rw username?            string
                              +--:(auth-type)
                                 +--:(password)
```
Internet-Draft      NETCONF Client and Server Models           June 2018

|     |     |        +--:(keystore)  
|     |     |                 {keystore-implemented)?  
|     |     |           +--rw reference  
|     |     |                   ks:asymmetric-key-ref  
|     |     +--:(certificate)  
|     |     |                   +--rw certificate {sshcmn:ssh-x509-cert}  
|     |     s)?  
|     |     +--rw (local-or-keystore)  
|     |     +--:(local)  
|     |     |        +--rw algorithm  
|     |     |        |       ct:key-algorithm-ref  
|     |     |        +--rw public-key  
|     |     |        |       binary  
|     |     |        +--rw private-key  
|     |     |        |       union  
|     |     |        +--rw cert  
|     |     |        |       ct:end-entity-cert-cms  
|     |     |     ---- n certificate-expiration  
|     |     |           +-- expiration-date?  
|     |     |           |       yang:date-and-time  
|     |     |     +--:(keystore)  
|     |     |                 {keystore-implemented)?  
|     |     |           +--rw reference  
|     |     |                   ks:asymmetric-key-cert  

ificate-ref  

|     |     |     +--rw server-auth  
|     |     |        +--rw pinned-ssh-host-keys?  
|     |     |        |       ta:pinned-host-keys-ref  
|     |     |        +--rw pinned-ca-certs?  
|     |     |        |       ta:pinned-certificates-ref  
|     |     |        |         {sshcmn:ssh-x509-certs}?  
|     |     |        +--rw pinned-server-certs?  
|     |     |        |       ta:pinned-certificates-ref  
|     |     |        |         {sshcmn:ssh-x509-certs}?  
|     |     |     +--rw transport-params  
|     |     |        |         {ssh-client-transport-params-config}?  
|     |     |        |             +--rw host-key  
|     |     |        |             |       +--rw host-key-alg* identityref  
|     |     |        |             +--rw key-exchange  
|     |     |        |             |       +--rw key-exchange-alg* identityref  
|     |     |        |             +--rw encryption  
|     |     |        |             |       +--rw encryption-alg* identityref  
|     |     |        |             +--rw mac  
|     |     |        |             |       +--rw mac-alg* identityref  
|     |     |     +--:(tls) {tls-listen)?  
|     |     |        +--rw tls  
|     |     |            +--rw address? inet:ip-address  

Watsen & Wu             Expires December 6, 2018                [Page 8]
3.2. Example Usage

The following example illustrates configuring a NETCONF client to initiate connections, using both the SSH and TLS transport protocols, as well as listening for call-home connections, again using both the SSH and TLS transport protocols.

This example is consistent with the examples presented in Section 2.2 of [I-D.ietf-netconf-keystore].

[Note: ‘\’ line wrapping for formatting only]
<netconf-client
xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-client">

<!-- NETCONF servers to initiate connections to -->
<initiate>
  <netconf-server>
    <name>corp-fw1</name>
    <endpoints>
      <endpoint>
        <name>corp-fw1.example.com</name>
        <ssh>
          <address>corp-fw1.example.com</address>
          <client-identity>
            <username>foobar</username>
            <public-key>
              <private-key>base64encodedvalue==</private-key>
            </public-key>
            <server-auth>
              <pinned-ca-certs>explicitly-trusted-server-ca-certs</pinned-ca-certs>
              <pinned-server-certs>explicitly-trusted-server-certs</pinned-server-certs>
            </server-auth>
          </client-identity>
        </ssh>
      </endpoint>
      <endpoint>
        <name>corp-fw2.example.com</name>
        <ssh>
          <address>corp-fw2.example.com</address>
          <client-identity>
            <username>foobar</username>
            <public-key>
              <private-key>base64encodedvalue==</private-key>
            </public-key>
            <server-auth>
              <pinned-ca-certs>explicitly-trusted-server-ca-certs</pinned-ca-certs>
              <pinned-server-certs>explicitly-trusted-server-certs</pinned-server-certs>
            </server-auth>
          </client-identity>
        </ssh>
      </endpoint>
    </endpoints>
  </netconf-server>
</initiate>

Watsen & Wu             Expires December 6, 2018               [Page 10]
3.3.  YANG Module

This YANG module has normative references to [RFC6242], [RFC6991], [RFC7589], [RFC8071], [I-D.ietf-netconf-ssh-client-server], and [I-D.ietf-netconf-tls-client-server].
yang-version 1.1;

prefix "ncc";

import ietf-inet-types {
  prefix inet;
  reference
    "RFC 6991: Common YANG Data Types";
}

import ietf-ssh-client {
  prefix ss;
  revision-date 2018-06-04; // stable grouping definitions
  reference
    "RFC YYYY: YANG Groupings for SSH Clients and SSH Servers";
}

import ietf-tls-client {
  prefix ts;
  revision-date 2018-06-04; // stable grouping definitions
  reference
    "RFC ZZZZ: YANG Groupings for TLS Clients and TLS Servers";
}

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

contact
  "WG Web:  <http://datatracker.ietf.org/wg/netconf/>
    WG List: <mailto:netconf@ietf.org>
    Author: Kent Watsen
           <mailto:kwatsen@juniper.net>
    Author: Gary Wu
           <mailto:garywu@cisco.com>";

description
  "This module contains a collection of YANG definitions for
configuring NETCONF clients.

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

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

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

revision "2018-06-04" {
  description
    "Initial version";
  reference
    "RFC XXXX: NETCONF Client and Server Models";
}

// Features

feature initiate {
  description
    "The ‘initiate’ feature indicates that the NETCONF client
    supports initiating NETCONF connections to NETCONF servers
    using at least one transport (e.g., SSH, TLS, etc.).";
}

feature ssh-initiate {
  description
    "The ‘ssh-initiate’ feature indicates that the NETCONF client
    supports initiating SSH connections to NETCONF servers.";
  reference
    "RFC 6242:
    Using the NETCONF Protocol over Secure Shell (SSH)";
}

feature tls-initiate {
  description
    "The ‘tls-initiate’ feature indicates that the NETCONF client
    supports initiating TLS connections to NETCONF servers.";
  reference
    "RFC 7589: Using the NETCONF Protocol over Transport
    Layer Security (TLS) with Mutual X.509
    Authentication";
}

feature listen {
  description
    "The ‘listen’ feature indicates that the NETCONF client
    supports opening a port to accept NETCONF server call
    home connections using at least one transport (e.g.,
feature ssh-listen {
    description
    "The 'ssh-listen' feature indicates that the NETCONF client
    supports opening a port to listen for incoming NETCONF
    server call-home SSH connections.";
    reference
    "RFC 8071: NETCONF Call Home and RESTCONF Call Home";
}

feature tls-listen {
    description
    "The 'tls-listen' feature indicates that the NETCONF client
    supports opening a port to listen for incoming NETCONF
    server call-home TLS connections.";
    reference
    "RFC 8071: NETCONF Call Home and RESTCONF Call Home";
}

container netconf-client {
    uses netconf-client;
    description
    "Top-level container for NETCONF client configuration.";
}

grouping netconf-client {
    description
    "Top-level grouping for NETCONF client configuration.";
}

container initiate {
    if-feature initiate;
    presence "Enables client to initiate TCP connections";
    description
    "Configures client initiating underlying TCP connections.";
    list netconf-server {
        key name;
        min-elements 1;
        description
        "List of NETCONF servers the NETCONF client is to
        initiate connections to in parallel.";
        leaf name {
            type string;
            description
            "An arbitrary name for the NETCONF server.";
        }
    }
    container endpoints {

Watsen & Wu Expires December 6, 2018 [Page 14]
description
"Container for the list of endpoints.";
list endpoint {
  key name;
  min-elements 1;
  ordered-by user;
  description
  "A user-ordered list of endpoints that the NETCONF client will attempt to connect to in the specified sequence. Defining more than one enables high-availability.";
  leaf name {
    type string;
    description
    "An arbitrary name for the endpoint.";
  }
  choice transport {
    mandatory true;
    description
    "Selects between available transports.";
    case ssh {
      if-feature ssh-initiate;
      container ssh {
        description
        "Specifies IP and SSH specific configuration for the connection.";
        leaf address {
          type inet:host;
          description
          "The IP address or hostname of the endpoint. If a domain name is configured, then the DNS resolution should happen on each usage attempt. If the DNS resolution results in multiple IP addresses, the IP addresses will be tried according to local preference order until a connection has been established or until all IP addresses have failed.";
        }
        leaf port {
          type inet:port-number;
          default 830;
          description
          "The IP port for this endpoint. The NETCONF client will use the IANA-assigned well-known port for 'netconf-ssh' (830) if no value is specified.";
        }
      }
      uses ss:ssh-client-grouping;
    }
}

case tls {
    if-feature tls-initiate;
    container tls {
        description
        "Specifies IP and TLS specific configuration for the connection.";
        leaf address {
            type inet:host;
            description
            "The IP address or hostname of the endpoint. If a domain name is configured, then the DNS resolution should happen on each usage attempt. If the DNS resolution results in multiple IP addresses, the IP addresses will be tried according to local preference order until a connection has been established or until all IP addresses have failed.";
        }
        leaf port {
            type inet:port-number;
            default 6513;
            description
            "The IP port for this endpoint. The NETCONF client will use the IANA-assigned well-known port for 'netconf-tls' (6513) if no value is specified.";
        }
        uses ts:tls-client-grouping {
            refine "client-identity/auth-type" {
                mandatory true;
                description
                "NETCONF/TLS clients MUST pass some authentication credentials.";
            }
        }
    }
}
} // end tls

container connection-type {
    description
    "Indicates the kind of connection to use.";
    choice connection-type {
        default persistent-connection;
description
"Selects between available connection types."

case persistent-connection {
  container persistent {
    presence
    "Indicates that a persistent connection is to be
    maintained."
    description
    "Maintain a persistent connection to the NETCONF
    server. If the connection goes down, immediately
    start trying to reconnect to it, using the
    reconnection strategy.
    This connection type minimizes any NETCONF server
to NETCONF client data-transfer delay, albeit at
the expense of holding resources longer."

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

    container keep-alives {
      description
      "Configures the keep-alive policy, to
      proactively test the aliveness of the SSH/TLS
      server. An unresponsive SSH/TLS server will
      be dropped after approximately max-attempts *
      max-wait seconds.";
      reference
      "RFC 8071: NETCONF Call Home and RESTCONF Call
      Home, Section 3.1, item S6";

      leaf max-wait {
        type uint16 {
          range "1..max"
        }
        units seconds;
        default 30;
        description
        "Sets the amount of time in seconds after
which if no data has been received from the
SSH/TLS server, a SSH/TLS-level message will
be sent to test the aliveness of the SSH/TLS
server.

leaf max-attempts {
  type uint8;
  default 3;
  description
    "Sets the maximum number of sequential keep-
    alive messages that can fail to obtain a
    response from the SSH/TLS server before
    assuming the SSH/TLS server is no longer
    alive."
}

case periodic-connection {
  container periodic {
    presence
      "Indicates that a periodic connection is to be
      maintained.";
    description
      "Periodically connect to the NETCONF server, so
      that the NETCONF server may deliver messages
      pending for the NETCONF client. The NETCONF
      server must close the connection when it is
      ready to release it. Once the connection has
      been closed, the NETCONF client will restart
      its timer until the next connection."
    leaf idle-timeout {
      type uint16;
      units "seconds";
      default 300; // five minutes
      description
        "Specifies the maximum number of seconds that
        a NETCONF session may remain idle. A NETCONF
        session will be dropped if it is idle for an
        interval longer than this number of seconds. If
        set to zero, then the server will never
        drop a session because it is idle. Sessions
        that have a notification subscription active
        are never dropped.";
    }
    leaf reconnect-timeout {
      type uint16 {
        range "1..max";
      }
    }
  }
}

Watsen & Wu
Expires December 6, 2018
[Page 18]
units minutes;
default 60;
description "Sets the maximum amount of unconnected time the NETCONF client will wait before re-establishing a connection to the NETCONF server. The NETCONF client may initiate a connection before this time if desired (e.g., to set configuration).";
}
}
}

container reconnect-strategy {
description "The reconnection strategy directs how a NETCONF client reconnects to a NETCONF server, after discovering its connection to the server has dropped, even if due to a reboot. The NETCONF client starts with the specified endpoint and tries to connect to it max-attempts times before trying the next endpoint in the list (round robin).";
leaf start-with {
type enumeration {
enum first-listed {
description "Indicates that reconnections should start with the first endpoint listed.";
}
enum last-connected {
description "Indicates that reconnections should start with the endpoint last connected to. If no previous connection has ever been established, then the first endpoint configured is used. NETCONF clients SHOULD be able to remember the last endpoint connected to across reboots.";
}
default first-listed;
description "Specifies which of the NETCONF server’s endpoints the NETCONF client should start with when trying to connect to the NETCONF server.";
}
leaf max-attempts {
type uint8 {
  range "1..max";
} default 3;

description
  "Specifies the number times the NETCONF client tries
  to connect to a specific endpoint before moving on
to the next endpoint in the list (round robin).";
}

list endpoint {
  key name;
  min-elements 1;
  description
    "List of endpoints to listen for NETCONF connections.";
  leaf name {
    type string;
    description
      "An arbitrary name for the NETCONF listen endpoint.";
  }
  choice transport {
    mandatory true;
    description
      "Selects between available transports.";
    case ssh {
      if-feature listen;
      presence "Enables client to accept call-home connections";
      description
        "Configures client accepting call-home TCP connections.";
      leaf idle-timeout {
        type uint16;
        units "seconds";
        default 3600; // one hour
        description
          "Specifies the maximum number of seconds that a NETCONF
          session may remain idle. A NETCONF session will be
dropped if it is idle for an interval longer than this
number of seconds. If set to zero, then the server
will never drop a session because it is idle. Sessions
that have a notification subscription active are never
dropped.";
      }
    }
}
} // end netconf-server
} // end initiate
if-feature ssh-listen;
container ssh {
  description
  "SSH-specific listening configuration for inbound
  connections.";
  leaf address {
    type inet:ip-address;
    description
    "The IP address to listen on for incoming call-
    home connections. The NETCONF client will listen
    on all configured interfaces if no value is
    specified. INADDR_ANY (0.0.0.0) or INADDR6_ANY
    (0:0:0:0:0:0:0:0 a.k.a. ::) MUST be used when
    the server is to listen on all IPv4 or IPv6
    addresses, respectively.";
  }
  leaf port {
    type inet:port-number;
    default 4334;
    description
    "The port number to listen on for call-home
    connections. The NETCONF client will listen
    on the IANA-assigned well-known port for
    'netconf-ch-ssh' (4334) if no value is
    specified.";
  }
  uses ss:ssh-client-grouping;
}
}
case tls {
  if-feature tls-listen;
  container tls {
    description
    "TLS-specific listening configuration for inbound
    connections.";
    leaf address {
      type inet:ip-address;
      description
      "The IP address to listen on for incoming call-
      home connections. The NETCONF client will listen
      on all configured interfaces if no value is
      specified. INADDR_ANY (0.0.0.0) or INADDR6_ANY
      (0:0:0:0:0:0:0:0 a.k.a. ::) MUST be used when
      the server is to listen on all IPv4 or IPv6
      addresses, respectively.";
    }
    leaf port {
      type inet:port-number;
      default 4334;
      description
      "The port number to listen on for call-home
      connections. The NETCONF client will listen
      on the IANA-assigned well-known port for
      'netconf-ch-ssl' (4334) if no value is
      specified.";
    }
  }
  uses ss:ssl-client-grouping;
}
4. The NETCONF Server Model

The NETCONF server model presented in this section supports servers both listening for connections as well as initiating call-home connections.

This model supports both the SSH and TLS transport protocols, using the SSH server and TLS server groupings defined in [I-D.ietf-netconf-ssh-client-server] and [I-D.ietf-netconf-tls-client-server] respectively.

All private keys and trusted certificates are held in the keystore model defined in [I-D.ietf-netconf-keystore].

YANG feature statements are used to enable implementations to advertise which parts of the model the NETCONF server supports.

4.1. Tree Diagram

The following tree diagram [RFC8340] provides an overview of the data model for the "ietf-netconf-server" module. Just the container is
displayed below, but there is also a reusable grouping by the same
name that the container is using.

[Note: \\
line wrapping for formatting only]

module: ietf-netconf-server
  +--rw netconf-server
    +--rw listen! {listen}?
      +--rw idle-timeout?  uint16
      +--rw endpoint* [name]
        +--rw name  string
        +--rw (transport)
          +--:(ssh) {ssh-listen}?
            +--rw ssh
              +--rw address  inet:ip-address
              +--rw port?   inet:port-number
            +--rw server-identity
              +--rw host-key* [name]
                +--rw name  string
                +--rw (host-key-type)
                  +--:(public-key)
                    +--rw public-key
                      +--rw (local-or-keystore)
                        +--:(local)
                          +--rw algorithm
crf
                          |       ct:key-algorithm-re\f
                      |       binary\f
                      +--rw private-key  union
                        +--:(keystore)
                          {keystore-implemented}?
                          +--rw reference
                          |   ks:asymmetric-key-r\ef
                        +--rw public-key  binary\ef
                        +--rw private-key

---rw encryption-alg* identityref
  +--rw mac
    +--rw mac-alg* identityref
  +--:(tls) {tls-call-home}?
    +--rw tls
      +--rw address inet:host
      +--rw port? inet:port-number
    +--rw server-identity
      +--rw (local-or-keystore)
        +--:(local)
          +--rw algorithm
            +--ct:key-algorithm-ref
          +--rw public-key
            +--binary
          +--rw private-key
            +--union
          +--rw cert
            +--ct:end-entity-cert-cms
            +--n certificate-expiration
              +-- expiration-date?
                yang:date-and-time
        +--:(keystore) {keystore-implemented?}
          +--rw reference
            ks:asymmetric-key-certifi cate-ref
          +--rw client-auth
            +--rw pinned-ca-certs?
              +--ta:pinned-certificates-ref
            +--rw pinned-client-certs?
              +--ta:pinned-certificates-ref
            +--rw cert-maps
              +--rw cert-to-name* [id]
                +--rw id uint32
                +--rw fingerprint
                  +--x509c2n:tls-fingerprint
                +--rw map-type identityref
                +--rw name string
            +--rw hello-params
              {tls-server-hello-params-config}?
                +--rw tls-versions
                  +--rw tls-version* identityref
                +--rw cipher-suites
                  +--rw cipher-suite* identityref
        +--rw connection-type
          +--rw (connection-type)?
            +-- persistent!
            +--rw (persistent-connection)
4.2. Example Usage

The following example illustrates configuring a NETCONF server to listen for NETCONF client connections using both the SSH and TLS transport protocols, as well as configuring call-home to two NETCONF clients, one using SSH and the other using TLS.

This example is consistent with the examples presented in Section 2.2 of [I-D.ietf-netconf-keystore].

[Note: ‘\’ line wrapping for formatting only]

```xml
<netconf-server
    xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server"
    xmlns:x509c2n="urn:ietf:params:xml:ns:yang:ietf-x509-cert-to-name">
  <!-- endpoints to listen for NETCONF connections on -->
  <listen>
    <endpoint> <!-- listening for SSH connections -->
      <name>netconf/ssh</name>
      <ssh>
        <address>192.0.2.7</address>
        <server-identity>
          <host-key>
            <name>deployment-specific-certificate</name>
            <public-key>
              <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-
              crypto-types">ct:secp521r1</algorithm>
              <private-key>base64encodedvalue==</private-key>
              <public-key>base64encodedvalue==</public-key>
            </public-key>
          </host-key>
        </server-identity>
        <client-cert-auth>
          ...
        </client-cert-auth>
      </ssh>
    </endpoint>
  </listen>
</netconf-server>
```
<pinned-ca-certs>explicitly-trusted-client-ca-certs</pinned-ca-certs>
<pinned-client-certs>explicitly-trusted-client-certs</pinned-client-certs>
</client-cert-auth>
</ssh>
</endpoint>
<endpoint> <!-- listening for TLS sessions -->
<name>netconf/tls</name>
<tls>
<address>192.0.2.7</address>
<server-identity>
to-types">ct:secp521r1</algorithm>
<private-key>base64encodedvalue==</private-key>
<public-key>base64encodedvalue==</public-key>
<cert>base64encodedvalue==</cert>
</server-identity>
</client-auth>
</tls>
</endpoint>
</listen>

<!-- calling home to SSH and TLS based NETCONF clients -->
<call-home>
<netconf-client> <!-- SSH-based client -->
<name>config-mgr</name>
<endpoints>
<endpoint>
<name>east-data-center</name>
<ssh>
  <address>east.config-mgr.example.com</address>
  <server-identity>
    <host-key>
      <name>deployment-specific-certificate</name>
      <public-key>
        <private-key>base64encodedvalue==</private-key>
        <public-key>base64encodedvalue==</public-key>
      </public-key>
    </host-key>
  </server-identity>
  <client-cert-auth>
    <pinned-ca-certs>explicitly-trusted-client-ca-certs</pinned-ca-certs>
    <pinned-client-certs>explicitly-trusted-client-certs</pinned-client-certs>
  </client-cert-auth>
</ssh>
</endpoint>

<endpoint>
  <name>west-data-center</name>
  <ssh>
    <address>west.config-mgr.example.com</address>
    <server-identity>
      <host-key>
        <name>deployment-specific-certificate</name>
        <public-key>
          <private-key>base64encodedvalue==</private-key>
          <public-key>base64encodedvalue==</public-key>
        </public-key>
      </host-key>
    </server-identity>
    <client-cert-auth>
      <pinned-ca-certs>explicitly-trusted-client-ca-certs</pinned-ca-certs>
      <pinned-client-certs>explicitly-trusted-client-certs</pinned-client-certs>
    </client-cert-auth>
  </ssh>
</endpoint>
</endpoints>

<connection-type>
  <periodic>
    <idle-timeout>300</idle-timeout>
  </periodic>
</connection-type>
<reconnect-timeout>60</reconnect-timeout>
</periodic>
</connection-type>
<reconnect-strategy>
<start-with>last-connected</start-with>
<max-attempts>3</max-attempts>
</reconnect-strategy>
</netconf-client>
<netconf-client> <!-- TLS-based client -->
<name>data-collector</name>
<endpoints>
<endpoint>
<name>east-data-center</name>
<tls>
<address>east.analytics.example.com</address>
<server-identity>
<private-key>base64encodedvalue==</private-key>
<public-key>base64encodedvalue==</public-key>
<cert>base64encodedvalue==</cert>
</server-identity>
<client-auth>
<pinned-ca-certs>explicitly-trusted-client-ca-certs</pinned-ca-certs>
<pinned-client-certs>explicitly-trusted-client-certs</pinned-client-certs>
<cert-maps>
<cert-to-name>
<id>1</id>
<fingerprint>11:0A:05:11:00</fingerprint>
<map-type>x509c2n:san-any</map-type>
</cert-to-name>
<cert-to-name>
<id>2</id>
<fingerprint>B3:4F:A1:8C:54</fingerprint>
<map-type>x509c2n:specified</map-type>
<name>scooby-doo</name>
</cert-to-name>
</cert-maps>
</client-auth>
</tls>
</endpoint>
<endpoint>
<name>west-data-center</name>
<tls>
<address>west.analytics.example.com</address>
<server-identity>

<private-key>base64encodedvalue==</private-key>

<pinned-ca-certs>
eneral-client-ca-certs
</pinned-ca-certs>

<pinned-client-certs>
eneral-client-certs
</pinned-client-certs>

<client-auth>

<pinned-ca-certs>explicitly-trusted-client-ca-certs</pinned-ca-certs>

<pinned-client-certs>explicitly-trusted-client-certs</pinned-client-certs>

<cert-maps>

<cert-to-name>
  <id>1</id>
  <fingerprint>11:0A:05:11:00</fingerprint>
  <map-type>x509c2n:san-any</map-type>
</cert-to-name>

<cert-to-name>
  <id>2</id>
  <fingerprint>B3:4F:A1:8C:54</fingerprint>
  <map-type>x509c2n:specified</map-type>
  <name>scooby-doo</name>
</cert-to-name>
</cert-maps>
</client-auth>

</tlsl>
</endpoints>

<connection-type>
  <persistent>
    <idle-timeout>300</idle-timeout>
    <keep-alives>
      <max-wait>30</max-wait>
      <max-attempts>3</max-attempts>
    </keep-alives>
  </persistent>
</connection-type>

<reconnect-strategy>
  <start-with>first-listed</start-with>
  <max-attempts>3</max-attempts>
</reconnect-strategy>

</netconf-client>
</call-home>
</netconf-server>
4.3. YANG Module

This YANG module has normative references to [RFC6242], [RFC6991],
[RFC7407], [RFC7589], [RFC8071],
[I-D.ietf-netconf-ssh-client-server], and
[I-D.ietf-netconf-tls-client-server].

This YANG module imports YANG types from [RFC6991], and YANG
groupings from [RFC7407], [I-D.ietf-netconf-ssh-client-server] and
[I-D.ietf-netconf-ssh-client-server].

<CODE BEGINS> file "ietf-netconf-server@2018-06-04.yang"
module ietf-netconf-server {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-server";
  prefix "ncs";

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-x509-cert-to-name {
    prefix x509c2n;
    reference
      "RFC 7407: A YANG Data Model for SNMP Configuration";
  }

  import ietf-ssh-server {
    prefix ss;
    revision-date 2018-06-04; // stable grouping definitions
    reference
      "RFC YYYY: YANG Groupings for SSH Clients and SSH Servers";
  }

  import ietf-tls-server {
    prefix ts;
    revision-date 2018-06-04; // stable grouping definitions
    reference
      "RFC ZZZZ: YANG Groupings for TLS Clients and TLS Servers";
  }

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

  contact

Watsen & Wu

Expires December 6, 2018
description
"This module contains a collection of YANG definitions for configuring NETCONF servers.

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

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

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

revision "2018-06-04" {
  description
    "Initial version";
  reference
    "RFC XXXX: NETCONF Client and Server Models";
}

// Features

feature listen {
  description
    "The ‘listen’ feature indicates that the NETCONF server supports opening a port to accept NETCONF client connections using at least one transport (e.g., SSH, TLS, etc.).";
}

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

feature tls-listen {
  description
  "The 'tls-listen' feature indicates that the NETCONF server supports opening a port to accept NETCONF over TLS client connections.";
  reference
  "RFC 7589: Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication";
}

feature call-home {
  description
  "The 'call-home' feature indicates that the NETCONF server supports initiating NETCONF call home connections to NETCONF clients using at least one transport (e.g., SSH, TLS, etc.).";
  reference
  "RFC 8071: NETCONF Call Home and RESTCONF Call Home";
}

feature ssh-call-home {
  description
  "The 'ssh-call-home' feature indicates that the NETCONF server supports initiating a NETCONF over SSH call home connection to NETCONF clients.";
  reference
  "RFC 8071: NETCONF Call Home and RESTCONF Call Home";
}

feature tls-call-home {
  description
  "The 'tls-call-home' feature indicates that the NETCONF server supports initiating a NETCONF over TLS call home connection to NETCONF clients.";
  reference
  "RFC 8071: NETCONF Call Home and RESTCONF Call Home";
}
container netconf-server {
    uses netconf-server;
    description
        "Top-level container for NETCONF server configuration."
}

grouping netconf-server {
    description
        "Top-level grouping for NETCONF server configuration.";
    container listen {
        if-feature listen;
        presence "Enables server to listen for TCP connections";
        description "Configures listen behavior";
        leaf idle-timeout {
            type uint16;
            units "seconds";
            default 3600; // one hour
            description
                "Specifies the maximum number of seconds that a NETCONF
                session may remain idle. A NETCONF session will be
dropped if it is idle for an interval longer than this
number of seconds. If set to zero, then the server
will never drop a session because it is idle. Sessions
that have a notification subscription active are never
dropped.";
        }
    }
    list endpoint {
        key name;
        min-elements 1;
        description
            "List of endpoints to listen for NETCONF connections.";
        leaf name {
            type string;
            description
                "An arbitrary name for the NETCONF listen endpoint.";
        }
    }
    choice transport {
        mandatory true;
        description
            "Selects between available transports.";
        case ssh {
            if-feature ssh-listen;
            container ssh {
                description
                    "SSH-specific listening configuration for inbound
                    connections.";
                leaf address {
                    type inet:ip-address;
                }
            }
        }
    }
}
mandatory true;

description
 "The IP address to listen on for incoming
 connections. The NETCONF server will listen
 on all configured interfaces if no value is
 specified. INADDR_ANY (0.0.0.0) or INADDR6_ANY
 (0:0:0:0:0:0:0:0 a.k.a. ::) MUST be used when
 the server is to listen on all IPv4 or IPv6
 addresses, respectively."

}

leaf port {
  type inet:port-number;
  default 830;
  description
  "The local port number to listen on. If no value
  is specified, the IANA-assigned port value for
  'netconf-ssh' (830) is used."
}

uses ss:ssh-server-grouping;

}

case tls {
  if-feature tls-listen;
  container tls {
    description
    "TLS-specific listening configuration for inbound
    connections."

    leaf address {
      type inet:ip-address;
      mandatory true;
      description
      "The IP address to listen on for incoming
      connections. The NETCONF server will listen
      on all configured interfaces if no value is
      specified. INADDR_ANY (0.0.0.0) or INADDR6_ANY
      (0:0:0:0:0:0:0:0 a.k.a. ::) MUST be used when
      the server is to listen on all IPv4 or IPv6
      addresses, respectively."
    }

    leaf port {
      type inet:port-number;
      default 6513;
      description
      "The local port number to listen on. If no value
      is specified, the IANA-assigned port value for
      'netconf-tls' (6513) is used."
    }

    uses ts:tls-server-grouping {

Watsen & Wu Expires December 6, 2018 [Page 37]
refine "client-auth" {
  must 'pinned-ca-certs or pinned-client-certs';
  description
    "NETCONF/TLS servers MUST validate client
certificates."
}

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

container call-home {
  if-feature call-home;
  presence "Enables server to initiate TCP connections";
  description "Configures call-home behavior";
  list netconf-client {
    key name;
    min-elements 1;
    description
      "List of NETCONF clients the NETCONF server is to
      initiate call-home connections to in parallel.";
    leaf name {
      type string;
      description
        "An arbitrary name for the remote NETCONF client.";
    }
  container endpoints {
    description
list endpoint {
    key name;
    min-elements 1;
    ordered-by user;
    description
        "A non-empty user-ordered list of endpoints for this
         NETCONF server to try to connect to in sequence.
         Defining more than one enables high-availability."
    leaf name {
        type string;
        description
            "An arbitrary name for this endpoint."
    }
    choice transport {
        mandatory true;
        description
            "Selects between available transports."
        case ssh {
            if-feature ssh-call-home;
            container ssh {
                description
                    "Specifies SSH-specific call-home transport
                     configuration."
                leaf address {
                    type inet:host;
                    mandatory true;
                    description
                        "The IP address or hostname of the endpoint.
                         If a domain name is configured, then the
                         DNS resolution should happen on each usage
                         attempt. If the the DNS resolution results
                         in multiple IP addresses, the IP addresses
                         will be tried according to local preference
                         order until a connection has been established
                         or until all IP addresses have failed."
                }
                leaf port {
                    type inet:port-number;
                    default 4334;
                    description
                        "The IP port for this endpoint. The NETCONF
                         server will use the IANA-assigned well-known
                         port for 'netconf-ch-ssh' (4334) if no value
                         is specified."
                }
            uses ss:ssh-server-grouping;
        }
    }
}
case tls {
    if-feature tls-call-home;
    container tls {
        description "Specifies TLS-specific call-home transport configuration."
        leaf address {
            type inet:host;
            mandatory true;
            description "The IP address or hostname of the endpoint. If a domain name is configured, then the DNS resolution should happen on each usage attempt. If the the DNS resolution results in multiple IP addresses, the IP addresses will be tried according to local preference order until a connection has been established or until all IP addresses have failed."
        }
        leaf port {
            type inet:port-number;
            default 4335;
            description "The IP port for this endpoint. The NETCONF server will use the IANA-assigned well-known port for 'netconf-ch-tls' (4335) if no value is specified."
        }
        uses ts:tls-server-grouping {
            refine "client-auth" {
                must 'pinned-ca-certs or pinned-client-certs';
                description "NETCONF/TLS servers MUST validate client certificates."
            }
            augment "client-auth" {
                description "Augments in the cert-to-name structure."
                container cert-maps {
                    uses x509c2n:cert-to-name;
                    description "The cert-maps container is used by a TLS-based NETCONF server to map the NETCONF client’s presented X.509 certificate to a NETCONF username. If no matching and valid cert-to-name list entry can be found, then the NETCONF"
server MUST close the connection, and
MUST NOT accept NETCONF messages over
it.";
reference
"RFC WWWW: NETCONF over TLS, Section 7";
}
}
}
} // end tls
} // end choice
} // end endpoint
}
}
}
} // end tls
} // end choice
} // end endpoint
}

container connection-type {

description
"Indicates the kind of connection to use.";

choice connection-type {

default persistent-connection;

description
"Selects between available connection types.";

case persistent-connection {

container persistent {

presence
"Indicates that a persistent connection is to be
maintained.";

description
"Maintain a persistent connection to the NETCONF
client. If the connection goes down, immediately
start trying to reconnect to it, using the
reconnection strategy.

This connection type minimizes any NETCONF client
to NETCONF server data-transfer delay, albeit at
the expense of holding resources longer.";

leaf idle-timeout {

type uint32;

units "seconds";

default 86400; // one day;

description
"Specifies the maximum number of seconds that
a NETCONF session may remain idle. A NETCONF
session will be dropped if it is idle for an
interval longer than this number of seconds.
If set to zero, then the server will never
drop a session because it is idle. Sessions
that have a notification subscription active
are never dropped.";

}
container keep-alives {
  description
    "Configures the keep-alive policy, to proactively test the aliveness of the SSH/TLS client. An unresponsive SSH/TLS client will be dropped after approximately max-attempts * max-wait seconds.";
  reference
    "RFC 8071: NETCONF Call Home and RESTCONF Call Home, Section 3.1, item S6";
  leaf max-wait {
    type uint16 {
      range "1..max";
    }
    units seconds;
    default 30;
    description
      "Sets the amount of time in seconds after which if no data has been received from the SSH/TLS client, a SSH/TLS-level message will be sent to test the aliveness of the SSH/TLS client.";
  }
  leaf max-attempts {
    type uint8;
    default 3;
    description
      "Sets the maximum number of sequential keep-alive messages that can fail to obtain a response from the SSH/TLS client before assuming the SSH/TLS client is no longer alive.";
  }
}
}

case periodic-connection {
  container periodic {
    presence
      "Indicates that a periodic connection is to be maintained.";
    description
      "Periodically connect to the NETCONF client, so that the NETCONF client may deliver messages pending for the NETCONF server. The NETCONF client must close the connection when it is ready to release it. Once the connection has been closed, the NETCONF server will restart
its timer until the next connection.

leaf idle-timeout {
  type uint16;
  units "seconds";
  default 300; // five minutes
  description
  "Specifies the maximum number of seconds that a NETCONF session may remain idle. A NETCONF session will be dropped if it is idle for an interval longer than this number of seconds. If set to zero, then the server will never drop a session because it is idle. Sessions that have a notification subscription active are never dropped."
}

leaf reconnect-timeout {
  type uint16 {
    range "1..max";
  }
  units minutes;
  default 60;
  description
  "Sets the maximum amount of unconnected time the NETCONF server will wait before re-establishing a connection to the NETCONF client. The NETCONF server may initiate a connection before this time if desired (e.g., to deliver an event notification message)."
}

}
}

container reconnect-strategy {
  description
  "The reconnection strategy directs how a NETCONF server reconnects to a NETCONF client, after discovering its connection to the client has dropped, even if due to a reboot. The NETCONF server starts with the specified endpoint and tries to connect to it max-attempts times before trying the next endpoint in the list (round robin)."
  leaf start-with {
    type enumeration {
      enum first-listed {
        description
        "Indicates that reconnections should start with the first endpoint listed."
      }
    }
  }
}
enum last-connected {
    description
    "Indicates that reconnections should start with the endpoint last connected to. If no previous connection has ever been established, then the first endpoint configured is used. NETCONF servers SHOULD be able to remember the last endpoint connected to across reboots.";
}

default first-listed;

description
    " Specifies which of the NETCONF client’s endpoints the NETCONF server should start with when trying to connect to the NETCONF client.";

leaf max-attempts {
    type uint8 {
        range "1..max";
    }
    default 3;

description
    " Specifies the number times the NETCONF server tries to connect to a specific endpoint before moving on to the next endpoint in the list (round robin).";
}

5. Design Considerations

Editorial: this section is a hold over from before, previously called "Objectives". It was only written two support the "server" (not the "client"). The question is if it’s better to add the missing "client" parts, or remove this section altogether.

The primary purpose of the YANG modules defined herein is to enable the configuration of the NETCONF client and servers. This scope includes the following objectives:
5.1. Support all NETCONF transports

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

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

5.2. Enable each transport to select which keys to use

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

5.3. Support authenticating NETCONF clients certificates

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

5.4. Support mapping authenticated NETCONF client certificates to usernames

When a client certificate is used for TLS client authentication, the NETCONF server must be able to derive a username from the authenticated certificate. Thus the modules defined herein should enable this mapping to be configured.

5.5. Support both listening for connections and call home

The NETCONF protocols were originally defined as having the server opening a port to listen for client connections. More recently the NETCONF working group defined support for call-home ([RFC8071]), enabling the server to initiate the connection to the client. Thus the modules defined herein should enable configuration for both listening for connections and calling home. Because implementations may not support both listening for connections and calling home, YANG "feature" statements should be used so that implementations can accurately advertise the connection types it supports.
5.6. For Call Home connections

The following objectives only pertain to call home connections.

5.6.1. Support more than one NETCONF client

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

5.6.2. Support NETCONF clients having more than one endpoint

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

5.6.3. Support a reconnection strategy

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

5.6.4. Support both persistent and periodic connections

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

5.6.5. Reconnection strategy for periodic connections

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

5.6.6. Keep-alives for persistent connections

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

5.6.7. Customizations for periodic connections

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

6. Security Considerations

The YANG module defined in this document uses groupings defined in [I-D.ietf-netconf-ssh-client-server] and [I-D.ietf-netconf-tls-client-server]. Please see the Security Considerations section in those documents for concerns related those groupings.

The YANG module defined in this document is designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:
The entire data trees defined by the modules defined in this draft are sensitive to write operations. For instance, the addition or removal of references to keys, certificates, trusted anchors, etc., can dramatically alter the implemented security policy. However, no NACM annotations are applied as the data SHOULD be editable by users other than a designated ‘recovery session’.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

NONE

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

NONE

7. IANA Considerations

7.1. The IETF XML Registry

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

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

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

7.2. The YANG Module Names Registry

This document registers two YANG modules in the YANG Module Names registry [RFC7950]. Following the format in [RFC7950], the following registrations are requested:
8. References

8.1. Normative References

[I-D.ietf-netconf-keystore]

[I-D.ietf-netconf-ssh-client-server]

[I-D.ietf-netconf-tls-client-server]


8.2. Informative References


Appendix A.  Change Log

A.1.  00 to 01
  o Renamed "keychain" to "keystore".

A.2.  01 to 02
  o Added to ietf-netconf-client ability to connect to a cluster of endpoints, including a reconnection-strategy.
  o Added to ietf-netconf-client the ability to configure connection-type and also keep-alive strategy.
  o Updated both modules to accommodate new groupings in the ssh/tls drafts.

A.3.  02 to 03
  o Refined use of tls-client-grouping to add a must statement indicating that the TLS client must specify a client-certificate.
  o Changed ‘netconf-client’ to be a grouping (not a container).

A.4.  03 to 04
  o Added RFC 8174 to Requirements Language Section.
  o Replaced refine statement in ietf-netconf-client to add a mandatory true.
  o Added refine statement in ietf-netconf-server to add a must statement.
  o Now there are containers and groupings, for both the client and server models.

A.5.  04 to 05
  o Now tree diagrams reference ietf-netmod-yang-tree-diagrams
  o Updated examples to inline key and certificates (no longer a leafref to keystore)
A.6.  05 to 06

- Fixed change log missing section issue.
- Updated examples to match latest updates to the crypto-types, trust-anchors, and keystore drafts.
- Reduced line length of the YANG modules to fit within 69 columns.

Acknowledgements

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

Authors’ Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Gary Wu
Cisco Networks

EMail: garywu@cisco.com
Abstract

This document provides a NETCONF binding to the dynamic subscription capability of both subscribed notifications and YANG push.

RFC Editor note: please replace the four references to pre-RFC normative drafts with the actual assigned RFC numbers.

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 February 4, 2019.

Copyright Notice

Copyright (c) 2018 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
1. Introduction

This document provides a binding for events streamed over the NETCONF protocol [RFC6241] for dynamic subscriptions as defined in [I-D.draft-ietf-netconf-subscribed-notifications]. In addition, as [I-D.ietf-netconf-yang-push] is itself built upon [I-D.draft-ietf-netconf-subscribed-notifications], this document enables a NETCONF client to request via a dynamic subscription and receive updates from a YANG datastore located on a NETCONF server.
2. 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are defined in [I-D.draft-ietf-netconf-subscribed-notifications]: notification message, event stream, publisher, receiver, subscriber, subscription.

3. Compatibility with RFC-5277’s create-subscription

A publisher is allowed to concurrently support dynamic subscription RPCs of [I-D.draft-ietf-netconf-subscribed-notifications] at the same time as [RFC5277]’s "create-subscription" RPC. However a single NETCONF transport session cannot support both this specification and a subscription established by [RFC5277]’s "create-subscription" RPC. To protect against any attempts to use a single NETCONF transport session in this way:

- A solution must reply with the [RFC6241] error "operation-not-supported" if a "create-subscription" RPC is received on a NETCONF session where an [I-D.draft-ietf-netconf-subscribed-notifications] established subscription exists.
- A solution must reply with the [RFC6241] error "operation-not-supported" if an "establish-subscription" request is been received on a NETCONF session where the "create-subscription" RPC has successfully [RFC5277] created a subscription.

If a publisher supports this specification but not subscriptions via [RFC5277], the publisher MUST NOT advertise "urn:ietf:params:netconf:capability:notification:1.0".

4. Mandatory XML, event stream and datastore support

The "encode-xml" feature of [I-D.draft-ietf-netconf-subscribed-notifications] is mandatory to support. This indicates that XML is a valid encoding for RPCs, state change notifications, and subscribed content.

A NETCONF publisher supporting event stream subscription via [I-D.draft-ietf-netconf-subscribed-notifications] MUST support the "NETCONF" event stream identified in that draft.
5.  NETCONF connectivity and the Dynamic Subscriptions

For a dynamic subscription, if the NETCONF session involved with the "establish-subscription" terminates, the subscription MUST be terminated.

For a dynamic subscription a "modify-subscription", "delete-subscription", or "resynch-subscription" RPC MUST be sent using same the NETCONF session upon which the referenced subscription was established.

6.  Notification Messages

Notification messages transported over the NETCONF protocol will use the "notification" message defined within [RFC5277], section 4.

For dynamic subscriptions, all notification messages MUST use the NETCONF transport session used by the "establish-subscription" RPC.

7.  Dynamic Subscriptions and RPC Error Responses

Management of dynamic subscriptions occurs via RPCs as defined in [I-D.ietf-netconf-yang-push] and [I-D.draft-ietf-netconf-subscribed-notifications]. When an RPC error occurs, the NETCONF RPC reply MUST include an "rpc-error" element per [RFC6241] with the error information populated as follows:

- an "error-type" node of "application".
- an "error-tag" node of "operation-failed".
- an "error-severity" of "error" (this MAY but does not have to be included).
- an "error-app-tag" node with the value being a string that corresponds to an identity associated with the error, as defined in [I-D.draft-ietf-netconf-subscribed-notifications] section 2.4.6 for general subscriptions, and [I-D.ietf-netconf-yang-push] Appendix A.1, for datastore subscriptions. The identityname to use depends on the RPC for which the error occurred. Viable errors for different RPCs are as follows:

<table>
<thead>
<tr>
<th>RPC</th>
<th>use base identity</th>
</tr>
</thead>
<tbody>
<tr>
<td>establish-subscription</td>
<td>establish-subscription-error</td>
</tr>
<tr>
<td>modify-subscription</td>
<td>modify-subscription-error</td>
</tr>
<tr>
<td>delete-subscription</td>
<td>delete-subscription-error</td>
</tr>
<tr>
<td>kill-subscription</td>
<td>kill-subscription-error</td>
</tr>
<tr>
<td>resynch-subscription</td>
<td>resynch-subscription-error</td>
</tr>
</tbody>
</table>
Each error identity will be inserted as the "error-app-tag" using JSON encoding following the form <modulename>:<identityname>. An example of such as valid encoding would be "ietf-subscribed-notifications:no-such-subscription".

- In case of error responses to an "establish-subscription" or "modify-subscription" request there is the option of including an "error-info" node. This node may contain XML-encoded data with hints for parameter settings that might lead to successful RPC requests in the future. Following are the yang-data structures which may be returned:

  - establish-subscription returns hints in yang-data structure
    \---------------------- ------------------------------------
    target: event stream   establish-subscription-stream-error-info
    target: datastore      establish-subscription-datastore-error-info

  - modify-subscription returns hints in yang-data structure
    \---------------------- ------------------------------------
    target: event stream   modify-subscription-stream-error-info
    target: datastore      modify-subscription-datastore-error-info

The yang-data included within "error-info" SHOULD NOT include the optional leaf "error-reason", as such a leaf would be redundant with information that is already placed within the "error-app-tag".

In case of an rpc error as a result of a "delete-subscription", a "kill-subscription", or a "resynch-subscription" request, no "error-info" needs to be included, as the "subscription-id" is the only RPC input parameter and no hints regarding this RPC input parameters need to be provided.

8. Security Considerations

If a malicious or buggy NETCONF subscriber sends a number of establish-subscription requests, then these subscriptions accumulate and may use up system resources. In such a situation, subscriptions MAY be terminated by terminating the underlying NETCONF session. The publisher MAY also suspend or terminate a subset of the active subscriptions on that NETCONF session.

9. Acknowledgments

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from: Andy Bierman, Yan Gang, Sharon Chisholm, Hector Trevino, Peipei Guo, Susan Hares, Tim Jenkins, Voit, et al.
10. Normative References

[I-D.draft-ietf-netconf-subscribed-notifications]

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


Appendix A. Examples
This section is non-normative.

A.1. Event Stream Discovery

As defined in [I-D.draft-ietf-netconf-subscribed-notifications] an event stream exposes a continuous set of events available for subscription. A NETCONF client can retrieve the list of available event streams from a NETCONF publisher using the "get" operation against the top-level container "/streams" defined in [I-D.draft-ietf-netconf-subscribed-notifications] Section 3.1.
The following example illustrates the retrieval of the list of available event streams:

```xml
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter type="subtree">
      <streams
    </filter>
  </get>
</rpc>
```

Figure 1: Get streams request

After such a request, the NETCONF publisher returns a list of event streams available, as well as additional information which might exist in the container.

A.2. Dynamic Subscriptions

A.2.1. Establishing Dynamic Subscriptions

The following figure shows two successful "establish-subscription" RPC requests as per [I-D.draft-ietf-netconf-subscribed-notifications]. The first request is given a subscription identifier of 22, the second, an identifier of 23.
To provide examples of the information being transported, example messages for interactions (a) and (b) in Figure 2 are detailed below:

```xml
<rpc message-id="102" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
    <stream>NETCONF</stream>
    <stream-xpath-filter xmlns:ex="http://example.com/events">
      /ex:foo/
    </stream-xpath-filter>
    <dscp>10</dscp>
  </establish-subscription>
</rpc>
```

Figure 3: establish-subscription request (a)
As NETCONF publisher was able to fully satisfy the request (a), the publisher sends the subscription identifier of the accepted subscription within message (b):

```
<rpc-reply message-id="102"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <identifier
xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
    22
  </identifier>
</rpc-reply>
```

Figure 4: establish-subscription success (b)

If the NETCONF publisher had not been able to fully satisfy the request, or subscriber has no authorization to establish the subscription, the publisher would have sent an RPC error response. For instance, if the "dscp" value of 10 asserted by the subscriber in Figure 3 proved unacceptable, the publisher may have returned:

```
<rpc-reply message-id="102"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>operation-failed</error-tag>
    <error-severity>error</error-severity>
    <error-app-tag>
      ietf-subscribed-notifications:dscp-unavailable
    </error-app-tag>
  </rpc-error>
</rpc-reply>
```

Figure 5: an unsuccessful establish subscription

The subscriber can use this information in future attempts to establish a subscription.

A.2.2. Modifying Dynamic Subscriptions

An existing subscription may be modified. The following exchange shows a negotiation of such a modification via several exchanges between a subscriber and a publisher. This negotiation consists of a failed RPC modification request/response, followed by a successful one.
If the subscription being modified in Figure 6 is a datastore subscription as per [I-D.ietf-netconf-yang-push], the modification request made in (c) may look like that shown in Figure 7. As can be seen, the modifications being attempted are the application of a new xpath filter as well as the setting of a new periodic time interval.

```xml
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <modify-subscription
    xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
    <identifier>23</identifier>
    <yp:datasource-xpath-filter xmlns="http://example.com/datasource">
      /interfaces-state/interface/oper-status
    </yp:datasource-xpath-filter>
    <yp:periodic>
      <yp:period>500</yp:period>
    </yp:periodic>
  </modify-subscription>
</rpc>
```

Figure 7: Subscription modification request (c)

If the NETCONF publisher can satisfy both changes, the publisher sends a positive result for the RPC. If the NETCONF publisher cannot
satisfy either of the proposed changes, the publisher sends an RPC error response (d). The following is an example RPC error response for (d) which includes a hint. This hint is an alternative time period value which might have resulted in a successful modification:

```xml
<rpc-reply message-id="303" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>operation-failed</error-tag>
    <error-severity>error</error-severity>
    <error-app-tag>
      ietf-yang-push:period-unsupported
    </error-app-tag>
    <error-info xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
      <modify-subscription-datastore-error-info>
        <period-hint>3000</period-hint>
      </modify-subscription-datastore-error-info>
    </error-info>
  </rpc-error>
</rpc-reply>
```

Figure 8: Modify subscription failure with Hint (d)

A.2.3. Deleting Dynamic Subscriptions

The following demonstrates deleting a subscription. This subscription may have been to either a stream or a datastore.

```xml
<rpc message-id="103" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <delete-subscription xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
    <identifier>22</identifier>
  </delete-subscription>
</rpc>
```

Figure 9: Delete subscription

If the NETCONF publisher can satisfy the request, the publisher replies with success to the RPC request.

If the NETCONF publisher cannot satisfy the request, the publisher sends an error-rpc element indicating the modification didn’t work. Figure 10 shows a valid response for existing valid subscription
identifier, but that subscription identifier was created on a different NETCONF transport session:

```xml
<rpc-reply message-id="103"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <rpc-error>
        <error-type>application</error-type>
        <error-tag>operation-failed</error-tag>
        <error-severity>error</error-severity>
        <error-app-tag>
            ietf-subscribed-notifications:no-such-subscription
        </error-app-tag>
    </rpc-error>
</rpc-reply>
```

Figure 10: Unsuccessful delete subscription

A.3. Subscription State Notifications

A publisher will send subscription state notifications for dynamic subscriptions according to the definitions within [I-D.draft-ietf-netconf-subscribed-notifications]).

A.3.1. subscription-modified

As per Section 2.7.2 of [I-D.draft-ietf-netconf-subscribed-notifications]), a "subscription-modified" might be sent if over NETCONF if the definition of a configured filter changes. A subscription state notification encoded in XML would look like:

```xml
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2007-09-01T10:00:00Z</eventTime>
    <subscription-modified
        xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
        <identifier>39</identifier>
            nsn:netconf
        </transport>
        <stream-xpath-filter xmlns:ex="http://example.com/events">
            /ex:foo
        </stream-xpath-filter>
    </subscription-modified>
</notification>
```

Figure 11: subscription-modified subscription state notification
A.3.2. subscription-resumed, and replay-complete

A "subscription-resumed" would look like:

```xml
<notification
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2007-09-01T10:00:00Z</eventTime>
    <subscription-resumed
        xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
        <identifier>39</identifier>
    </subscription-resumed>
</notification>
```

Figure 12: subscription-resumed notification in XML

The "replay-complete" is virtually identical, with "subscription-resumed" simply being replaced by "replay-complete".

A.3.3. subscription-terminated and subscription-suspended

A "subscription-terminated" would look like:

```xml
<notification
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2007-09-01T10:00:00Z</eventTime>
    <subscription-terminated
        xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
        <identifier>39</identifier>
        <error-id>
            suspension-timeout
        </error-id>
    </subscription-terminated>
</notification>
```

Figure 13: subscription-terminated subscription state notification

The "subscription-suspended" is virtually identical, with "subscription-terminated" simply being replaced by "subscription-suspended".

Appendix B. Changes between revisions

(To be removed by RFC editor prior to publication)
B.1. v10 to v11
  o Configured removed.

B.2. v09 to v10
  o Tweaks to examples and text.
  o Downshifted state names.
  o Removed address from examples.

B.3. v08 to v09
  o Tweaks based on Kent’s comments.
  o Updated examples in Appendix A. And updates to some object names based on changes in the subscribed-notifications draft.
  o Added a YANG model for the NETCONF identity.

B.4. v07 to v08
  o Tweaks and clarification on :interleave.

B.5. v06 to v07
  o XML encoding and operational datastore mandatory.
  o Error mechanisms and examples updated.

B.6. v05 to v06
  o Moved examples to appendices
  o All examples rewritten based on namespace learnings
  o Normative text consolidated in front
  o Removed all mention of JSON
  o Call home process detailed
  o Note: this is a major revision attempting to cover those comments received from two week review.

B.7. v03 to v04
  o Added additional detail to "configured subscriptions"
  o Added interleave capability
  o Adjusted terminology to that in draft-ietf-netconf-subscribed-notifications
  o Corrected namespaces in examples
B.8. v01 to v03
  o Text simplifications throughout
  o v02 had no meaningful changes

B.9. v00 to v01
  o Added Call Home in solution for configured subscriptions.
  o Clarified support for multiple subscription on a single session.
     No need to support multiple create-subscription.
  o Added mapping between terminology in yang-push and [RFC6241] (the
     one followed in this document).
  o Editorial improvements.

Authors’ Addresses

  Eric Voit
  Cisco Systems
  Email: evoit@cisco.com

  Alexander Clemm
  Huawei
  Email: ludwig@clemm.org

  Alberto Gonzalez Prieto
  Microsoft
  Email: alberto.gonzalez@microsoft.com

  Einar Nilsen-Nygaard
  Cisco Systems
  Email: einarnn@cisco.com

  Ambika Prasad Tripathy
  Cisco Systems
  Email: ambtripa@cisco.com
NETCONF Extensions to Support the Network Management Datastore Architecture
draft-ietf-netconf-nmda-netconf-06

Abstract

This document extends the NETCONF protocol defined in RFC 6241 in order to support the Network Management Datastore Architecture defined in RFC 8342.

This document updates both RFC 6241 and RFC 7950. The update to RFC 6241 adds new operations <get-data> and <edit-data>, and augments existing operations <lock>, <unlock>, and <validate>. The update to RFC 7950 requires the usage of I-D.ietf-netconf-rfc7895bis by NETCONF servers implementing the Network Management Datastore Architecture.

RFC Ed.: Please replace "I-D.ietf-netconf-rfc7895bis" above with its final RFC assignment and remove this note.

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 November 29, 2018.
1. Introduction

This document extends the NETCONF protocol defined in [RFC6241] in order to support the Network Management Datastore Architecture (NMDA) defined in [RFC8342].

This document updates [RFC6241] in order to enable NETCONF clients to interact with all the datastores supported by a server implementing the NMDA. The update both adds new operations <get-data> and <edit-data>, and augments existing operations <lock>, <unlock>, and <validate>.

This document also updates [RFC7950] in order to enable NETCONF clients to both discover which datastores are supported by the...
NETCONF server, as well as determine which modules are supported in each datastore. The update requires NETCONF servers implementing the NMDA to support [I-D.ietf-netconf-rfc7895bis].

1.1. Terminology

This document uses the terminology defined by the NMDA [RFC8342]. The following term is defined in [I-D.ietf-netconf-rfc7895bis]:

- YANG library checksum

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.2. Tree Diagrams

Tree diagrams used in this document follow the notation defined in [RFC8340].

2. Datastore and YANG Library Requirements

RFC Ed.: Update 201X-XX-XX below with correct date.

An NMDA-compliant NETCONF server MUST support the operational state datastore and it MUST implement at least revision 201X-XX-XX of the "ietf-yang-library" module defined in [I-D.ietf-netconf-rfc7895bis].

A NETCONF client can discover which datastores and YANG modules the server supports by reading the YANG library information from the operational state datastore.

The server MUST advertise the following capability in the <hello> message (line breaks and whitespaces are used for formatting reasons only):

```
urn:ietf:params:netconf:capability:yang-library:1.1?
  revision=<date>&checksum=<checksum-value>
```

The parameter "revision" has the same value as the revision date of the "ietf-yang-library" module implemented by the server. This parameter MUST be present.

The parameter "checksum" contains the YANG library checksum [I-D.ietf-netconf-rfc7895bis]. This parameter MUST be present.
With this mechanism, a client can cache the supported datastores and YANG modules for a server and only update the cache if the "checksum" value in the <hello> message changes.

This document updates [RFC7950], Section 5.6.4, to allow servers to advertise the capability :yang-library:1.1 instead of :yang-library:1.0, and to implement the subtree "/yang-library" [I-D.ietf-netconf-rfc7895bis] instead of "/modules-state".

3. NETCONF Extensions

This section describes the NETCONF extensions needed to support the NMDA. These changes are defined in a new YANG ([RFC7950]) module "ietf-netconf-nmda".

These changes include the use of source and target parameters based on the "datastore" identity defined in the "ietf-datastores" module [RFC8342]. The use of identities allows future expansion in a way that the choice-based strategy from the original operations (e.g., <get-config>, <edit-config>) does not.

3.1. New NETCONF Operations

Two new operations <get-data> and <edit-data> are defined in this document in order to support the NMDA. These operations are similar to the <get-config> and <edit-config> operations but they can work on an extensible set of datastores.

3.1.1. The <get-data> Operation

The <get-data> operation retrieves data from a specific NMDA datastore. This operation is similar to NETCONF’s <get-config> operation defined in [RFC6241], but it adds the flexibility to select the source datastore.
The "datastore" parameter indicates the datastore which is the source of the data to be retrieved. This is a datastore identity.

The <get-data> operation accepts a content filter parameter, similar to the "filter" parameter of <get-config>, but using explicit nodes for subtree filtering ("subtree-filter") and XPath filtering ("xpath-filter").

The "config-filter" parameter can be used to retrieve only "config true" or "config false" nodes.

The "origin-filter" parameter, which can be present multiple times, selects nodes matching any of the given values. The "negated-origin-filter", which can be present multiple times, selects nodes that do not match all given values. The "origin-filter" and "negated-origin-filter" parameters cannot be used together.

The "max-depth" parameter can be used by the client to limit the number of sub-tree levels that are returned in the reply.

3.1.1.1. With-defaults interactions

If the "with-defaults" capability is supported by the server, then the "with-defaults" parameter, defined in [RFC6243], is supported for <get-data> operations that target conventional configuration datastores.
The "with-defaults" parameter is optional to support for <get-data> operations that target <operational>. The associated capability to indicate a server’s support is identified with the URI:

urn:ietf:params:netconf:capability:with-operational-defaults:1.0

If the "with-defaults" parameter is supported for <get-data> operations on <operational>, then all retrieval modes specified in either the 'basic-mode' or 'also-supported' parameters of the "with-defaults" capability are permitted. The behavior of the "with-defaults" parameter for <operational> is defined as below:

- If no "with-defaults" parameter is specified, or if it is set to "explicit", "report-all", or "report-all-tagged", then the "in use" values, as defined in [RFC8342] section 5.3, are returned from the operational state datastore, even if a node happens to have a default statement in the YANG module, and this default value is being used by the server. If the "with-defaults" parameter is set to "report-all-tagged", any values that match the schema default are tagged with additional metadata, as described in [RFC6243] section 3.4.

- If the "with-defaults" parameter is set to "trim", all "in use" values are returned, except that the output is filtered to exclude any values that match the default defined in the YANG schema.

Support for "with-defaults" in <get-data> operations on any datastore not defined in [RFC8342] SHOULD be defined by the specification for the datastore.

3.1.1.2. Origin Metadata Attribute

The <get-data> operation defines a parameter named "with-origin", which if present, requests that the server includes "origin" metadata annotations in its response, as detailed in the NMDA. This parameter is only valid for the operational state datastore and any datastores with identities derived from the "operational" identity. Otherwise, if an invalid datastore is specified then an error is returned, as specified in "ietf-netconf-nmda" (see Section 4). Note that "origin" metadata annotations are not included in a response unless a client explicitly requests them.

Data in the operational state datastore can come from multiple sources. The server should return the most accurate value for the "origin" metadata annotation as possible, indicating the source of the operational value, as specified in Section 5.3.4 of [RFC8342].
When encoding the origin metadata annotation for a hierarchy of returned nodes, the annotation may be omitted for a child node when the value matches that of the parent node, as described in the "ietf-origin" YANG module [RFC8342].

The "with-origin" parameter is optional to support. It is identified with the feature "origin".

3.1.1.3. Example: Retrieving an entire subtree from <running>

The following example shows the <get-data> version of the <get-config> example shown in Section 7.1 of [RFC6241].

```xml
<rpc message-id="101"
 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <datastore>ds:running</datastore>
   <subtree-filter>
     <top xmlns="http://example.com/schema/1.2/config">
       <users/>
     </top>
   </subtree-filter>
 </get-data>
</rpc>

<rpc-reply message-id="101"
 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-nmda">
   <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>
```
3.1.2. The <edit-data> Operation

The <edit-data> operation changes the contents of a writable datastore, similar to the <edit-config> operation defined in [RFC6241], but with additional flexibility in naming the target datastore. If an <edit-data> operation is invoked on a non-writable datastore, then an error is returned, as specified in "ietf-netconf-nmda" (see Section 4).

```
+-x edit-data
  +---w input
    +----w datastore          ds:datastore-ref
    +----w default-operation? enumeration
    +----w (edit-content)
      +------:(config)
        +----w config?   <anydata>
      +------:(url)
        +----w url?       inet:uri {nc:url}?{2}
```

The "datastore" parameter is a datastore identity that indicates the desired target datastore where changes should be made.

The "default-operation" parameter is a copy of the "default-operation" parameter of the <edit-config> operation.

The "edit-content" choice mirrors the "edit-content" choice of the <edit-config> operation. Note, however, that the "config" element in the "edit-content" choice of <edit-data> uses "anydata" (introduced in YANG 1.1) while the "config" element in the "edit-content" choice of <edit-config> used "anyxml".

The <edit-data> operation does not support the "error-option" and the "test-option" parameters that were part of the <edit-config> operation. The error behaviour of <edit-data> corresponds to the "error-option" "rollback-on-error".

If the "with-defaults" capability is supported by the server, the semantics of editing modes is the same as for <edit-config>, as described in section 4.5.2 of [RFC6243].

Semantics for "with-defaults" in <edit-data> operations on any non-conventional configuration datastores SHOULD be defined by the specification for the datastore.
3.1.2.1. Example: Setting a leaf of an interface in <running>

The following example shows the <edit-data> version of the first <edit-config> example in Section 7.2 of [RFC6241], setting the MTU to 1500 on an interface named "Ethernet0/0" in the running configuration datastore.

```
<rpc message-id="102"
xmllns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <datastore>ds:running</datastore>
    <config>
      <top xmlns="http://example.com/schema/1.2/config">
        <interface>
          <name>Ethernet0/0</name>
          <mtu>1500</mtu>
        </interface>
      </top>
    </config>
  </edit-data>
</rpc>

<rpc-reply message-id="102"
xmllns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

The other <edit-config> examples shown in Section 7.2 can be translated to <edit-data> examples in a similar way.

3.2. Augmentations to NETCONF Operations

Several of the operations defined in the base NETCONF YANG module "ietf-netconf" [RFC6241] may be used with new datastores. Hence, the <lock>, <unlock>, and <validate> operations are augmented with a new "datastore" leaf that can select the desired datastore. If a <lock>, <unlock>, or <validate> operation is not supported on a particular datastore then an error is returned, as specified in "ietf-netconf-nmda" (see Section 4).

4. NETCONF Datastores YANG Module

This module imports definitions from [RFC6991], [RFC6241], [RFC6243], and [RFC8342].
module ietf-netconf-nmda {
    yang-version 1.1;
    prefix ncds;

    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-datastores {
        prefix ds;
        reference "RFC 8342: Network Management Datastore Architecture.";
    }

    import ietf-origin {
        prefix or;
        reference "RFC 8342: Network Management Datastore Architecture.";
    }

    import ietf-netconf {
        prefix nc;
        reference "RFC 6241: Network Configuration Protocol (NETCONF)";
    }

    import ietf-netconf-with-defaults {
        prefix ncwd;
        reference "RFC 6243: With-defaults Capability for NETCONF.";
    }

    organization
        "IETF NETCONF Working Group";
    contact
        "WG Web:  <https://datatracker.ietf.org/wg/netconf/>
        WG List:  <mailto:netconf@ietf.org>
        Author:   Martin Bjorklund
                  <mailto:mbj@tail-f.com>
        Author:   Juergen Schoenwaelder
                  <mailto:j.schoenwaelder@jacobs-university.de>
This YANG module defines a set of NETCONF operations to support the Network Management Datastore Architecture (NMDA).

Copyright (c) 2018 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 (http://www.rfc-editor.org/info/rfcxxxx); see the RFC itself for full legal notices."

// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
// RFC Ed.: replace XXXX with actual RFC number and remove this
// note.
revision 2018-05-22 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: NETCONF Extensions to Support the Network Management Datastore Architecture";
}

feature origin {
  description
    "Indicates that the server supports the 'origin' annotation.";
  reference
    "RFC 8342: Network Management Datastore Architecture";
}

feature with-defaults {
  description
    "NETCONF :with-defaults capability; If the server advertises
the :with-defaults capability for a session, then this
feature must also be enabled for that session. Otherwise,
this feature must not be enabled.

reference
"RFC 6243: With-defaults Capability for NETCONF, section 4; and
RFC XXXX: NETCONF Extensions to Support the Network Management
Datastore Architecture, section 3.1.1.1."

rpc get-data {
  description
  "Retrieve data from an NMDA datastore. The content returned
  by get-data must satisfy all filters, i.e., the filter
criteria are logically ANDed.

  The 'with-origin' parameter is only valid for an operational
datastore. If 'with-origin' is used with an invalid datastore,
then the server MUST return an <rpc-error> element with an
<error-tag> value of 'invalid-value'.

  The 'with-defaults' parameter only applies to the operational
datastore if the NETCONF :with-defaults and
:with-operational-defaults capabilities are both advertised.
If the 'with-defaults' parameter is present in a request for
which it is not supported, then the server MUST return an
<rpc-error> element with an <error-tag> value of
'invalid-value'."

  input {
    leaf datastore {
      type ds:datastore-ref;
      mandatory true;
      description
      "Datastore from which to retrieve data.

      If the datastore is not supported by the server, then the
      server MUST return an <rpc-error> element with an
      <error-tag> value of 'invalid-value'."
    }
  }

  choice filter-spec {
    description
    "The content filter specification for this request."
  }

  anydata subtree-filter {
    description
    "This parameter identifies the portions of the
target datastore to retrieve."
  }
}
leaf xpath-filter {
    if-feature nc:xpath;
    type yang:xpath1.0;
    description
        "This parameter contains an XPath expression identifying
        the portions of the target datastore to retrieve.

        If the expression returns a node-set, all nodes in the
        node-set are selected by the filter. Otherwise, if the
        expression does not return a node-set, then the get-data
        operation fails.

        The expression is evaluated in the following XPath
        context:

        o  The set of namespace declarations are those in
        scope on the 'xpath-filter' leaf element.

        o  The set of variable bindings is empty.

        o  The function library is the core function library,
        and the XPath functions defined in section 10 in
        RFC 7950.

        o  The context node is the root node of the target
        datastore.";
}

leaf config-filter {
    type boolean;
    description
        "Filter for nodes with the given value for their
        'config' property."
}

choice origin-filters {
    when 'derived-from-or-self(datastore, "ds:operational")';
    if-feature origin;
    description
        "Filters based on the 'origin' annotation.";

    leaf-list origin-filter {
        type or:origin-ref;
        description
            "Filter based on the 'origin' annotation. A node matches
the filter if its 'origin' annotation is derived from or
equal to any of the given filter values.

leaf-list negated-origin-filter {
  type or:origin-ref;
  description
  "Filter based on the 'origin' annotation. A node matches
  the filter if its 'origin' annotation is not derived
  from and not equal to all of the given filter values.";
}

leaf max-depth {
  type union {
    type uint16 {
      range "1..65535";
    }
    type enumeration {
      enum "unbounded" {
        description
        "All descendant nodes are included.";
      }
    }
  }
  default "unbounded";
  description
  "For each node selected by the filter, this parameter
  selects how many conceptual sub-tree levels should be
  returned in the reply. If the depth is 1, the reply
  includes just the selected nodes but no children. If the
  depth is 'unbounded', all descendant nodes are included.";
}

leaf with-origin {
  when 'derived-from-or-self(../datastore, "ds:operational")';
  if-feature origin;
  type empty;
  description
  "If this parameter is present, the server will return
  the 'origin' annotation for the nodes that has one.";
}

uses ncwd:with-defaults-parameters {
  if-feature with-defaults;
}

output {

anydata data {
  description
  "Copy of the source datastore subset which matched
  the filter criteria (if any). An empty data
  container indicates that the request did not
  produce any results.";
}
}
}

cpy edit-data {
  description
  "Edit data in an NMDA datastore.

  If an error condition occurs such that an error severity
  <rpc-error> element is generated, the server will stop
  processing the <edit-data> operation and restore the
  specified configuration to its complete state at
  the start of this <edit-data> operation.";
  input {
    leaf datastore {
      type ds:datastore-ref;
      mandatory true;
      description
      "Datastore which is the target of the edit-data operation.

      If the target datastore is not writable, or is not
      supported by the server, then the server MUST return an
      <rpc-error> element with an <error-tag> value of
      'invalid-value'.";
    }
    leaf default-operation {
      type enumeration {
        enum "merge" {
          description
          "The default operation is merge.";
        }
        enum "replace" {
          description
          "The default operation is replace.";
        }
        enum "none" {
          description
          "There is no default operation.";
        }
      }
      default "merge";
      description
      "The default operation is merge.";
    }
  }
}
"The default operation to use."

choice edit-content {
  mandatory true;
  description "The content for the edit operation."
}

anydata config {
  description "Inline config content."
}

leaf url {
  if-feature nc:url;
  type inet:uri;
  description "URL based config content."
}

augment "/nc:lock/nc:input/nc:target/nc:config-target" {
  description "Add NMDA Datastore as target."
  leaf datastore {
    type ds:datastore-ref;
    description "Datastore to lock."

    The lock operation is only supported on writable datastores.

    If the lock operation is not supported by the server on the
    specified target datastore, then the server MUST return an
    <rpc-error> element with an <error-tag> value of
    'invalid-value'."
  }
}

augment "/nc:unlock/nc:input/nc:target/nc:config-target" {
  description "Add NMDA Datastore as target."
  leaf datastore {
    type ds:datastore-ref;
    description "Datastore to unlock."

    The unlock operation is only supported on writable datastores.

    If the unlock operation is not supported by the server on the
    specified target datastore, then the server MUST return an
    <rpc-error> element with an <error-tag> value of
    'invalid-value'."
  }
}
"Datastore to unlock.

The unlock operation is only supported on writable datastores.

If the unlock operation is not supported by the server on the specified target datastore, then the server MUST return an <rpc-error> element with an <error-tag> value of 'invalid-value'.

*/

augment "/nc:validate/nc:input/nc:source/nc:config-source" {
  description
    "Add NMDA Datastore as source.";
  leaf datastore {
    type ds:datastore-ref;
    description
      "Datastore to validate.

The validate operation is supported only on configuration datastores.

If the validate operation is not supported by the server on the specified target datastore, then the server MUST return an <rpc-error> element with an <error-tag> value of 'invalid-value'.";
  }
}

5. IANA Considerations

This document registers two capability identifier URNs in the "Network Configuration Protocol (NETCONF) Capability URNs" registry:
Index

Capability Identifier
-----------------------
yang-library
urn:ietf:params:netconf:capability:yang-library:1.1

:with-operational-defaults
urn:ietf:params:netconf:capability:with-operational-defaults:1.0

This document registers a URI in the "IETF XML Registry" [RFC3688].
Following the format in RFC 3688, the following registration has been made.

Registrant Contact: The IESG.
XML:    N/A, the requested URI is an XML namespace.

This document registers a YANG module in the "YANG Module Names" registry [RFC6020].

name:         ietf-netconf-nmda
prefix:       ncds
reference:    RFC XXXX

6. Security Considerations

The YANG module defined in this document extends the base operations of the NETCONF [RFC6241] protocol. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242].

The network configuration access control model [RFC8341] provides the means to restrict access for particular NETCONF users to a preconfigured subset of all available NETCONF protocol operations and content.

The security considerations for the base NETCONF protocol operations (see Section 9 of [RFC6241]) apply to the new NETCONF <get-data> and <edit-data> operations defined in this document.

7. References
7.1. Normative References

[I-D.ietf-netconf-rfc7895bis]


7.2. Informative References


Authors' Addresses

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

Juergen Schoenwaelder  
Jacobs University  
Email: j.schoenwaelder@jacobs-university.de

Phil Shafer  
Juniper Networks  
Email: phil@juniper.net

Kent Watsen  
Juniper Networks  
Email: kwatsen@juniper.net

Robert Wilton  
Cisco Systems  
Email: rwilton@cisco.com
RESTCONF Extensions to Support the Network Management Datastore Architecture
draft-ietf-netconf-nmda-restconf-04

Abstract

This document extends the RESTCONF protocol defined in RFC 8040 in order to support the Network Management Datastore Architecture defined in RFC 8342.

This document updates RFC 8040 by introducing new datastore resources, adding a new query parameter, and requiring the usage of I-D.ietf-netconf-rfc7895bis by RESTCONF servers implementing the Network Management Datastore Architecture.

RFC Ed.: Please replace "I-D.ietf-netconf-rfc7895bis" above with its final RFC assignment and remove this note.

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 22, 2018.
1. Introduction

This document extends the RESTCONF protocol defined in [RFC8040] in order to support the Network Management Datastore Architecture (NMDA) defined in [RFC8342].

This document updates [RFC8040] in order to enable RESTCONF clients to discover which datastores are supported by the RESTCONF server, determine which modules are supported in each datastore, and to interact with all the datastores supported by the NMDA. Specifically, the update introduces new datastore resources, adds a new query parameter, and requires the usage of [I-D.ietf-netconf-rfc7895bis] by RESTCONF servers implementing the NMDA.
The solution presented in this document is backwards compatible with [RFC8040]. This is achieved by only adding new resources and leaving the semantics of the existing resources unchanged.

1.1. Terminology

This document uses the terminology defined by the NMDA [RFC8342].

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Datastore and YANG Library Requirements

RFC Ed.: Please update 201X-XX-XX below with correct date and remove this note.

An NMDA-compliant RESTCONF server MUST support the operational state datastore and it MUST implement at least revision 201X-XX-XX of the "ietf-yang-library" module defined in [I-D.ietf-netconf-rfc7895bis].

Such a server identifies that it supports the NMDA both by implementing the {+restconf}/ds/ietf-datastores:operational resource, and by implementing at least revision 201X-XX-XX of the "ietf-yang-library" module.

A RESTCONF client can test if a server supports the NMDA by using either the HEAD or GET methods on {+restconf}/ds/ietf-datastores:operational.

A RESTCONF client can discover which datastores and YANG modules the server supports by reading the YANG library information from the operational state datastore.

3. RESTCONF Extensions

This section describes the RESTCONF extensions needed to support the NMDA.

3.1. New Datastore Resources

This document defines a set of new resources representing datastores as defined in [RFC8342]. These resources are available using the following resource path template:

{+restconf}/ds/<datastore>
The <datastore> path component is encoded as an "identity" according to the JSON encoding rules for identities, defined in Section 4 of [RFC7951]. Such an identity MUST be derived from the "datastore" identity defined in the "ietf-datastores" YANG module [RFC8342].

Specifically:

- The resource `{+restconf}/ds/ietf-datastores:operational` refers to the operational state datastore.
- The resource `{+restconf}/ds/ietf-datastores:running` refers to the running configuration datastore.
- The resource `{+restconf}/ds/ietf-datastores:intended` refers to the intended configuration datastore.

An NMDA-compliant server MUST implement `{+restconf}/ds/ietf-datastores:operational`. Other datastore resources are optional to implement.

YANG actions can only be invoked in `{+restconf}/ds/ietf-datastores:operational`.

If a server implements the example datastore "ds-ephemeral" in the module "example-ds-ephemeral", it would implement the resource `{+restconf}/ds/example-ds-ephemeral:ds-ephemeral`.

3.2. Protocol Operations

The protocol operations available for the new datastore resources (Section 3.1) are the same as the protocol operations defined in [RFC8040] for the `{+restconf}/data` resource with the following exceptions:

- Dynamic configuration datastores are excluded, as each dynamic configuration datastore definition needs to be reviewed for what protocol operations it supports.
- Some datastores are read-only by nature (e.g., <intended>), and hence any attempt to modify these datastores will fail. A server MUST return a response with a "405 Method Not Allowed" status-line and error-tag value "operation-not-supported".
- The semantics of the "with-defaults" query parameter ([RFC8040], Section 4.8.9) differs when interacting with the operational state datastore. The semantics are described below, in Section 3.2.1.
The "with-defaults" query parameter ([RFC8040], Section 4.8.9) is optional to support when interacting with {+restconf}/ds/ietf-datastores:operational. The associated capability to indicate a server's support is identified with the URI:

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

For servers that support it, the behavior of the "with-defaults" query parameter on the operational state datastore is defined as follows:

- If no "with-defaults" query parameter is specified, or if it is set to "explicit", "report-all", or "report-all-tagged", then the "in use" values, as defined in [RFC8342] section 5.3, are returned from the operational state datastore, even if a node happens to have a default statement in the YANG module and this default value is being used by the server. If the "with-defaults" parameter is set to "report-all-tagged", any values that match the schema default are tagged with additional metadata, as described in [RFC8040], Section 4.8.9.

- If the "with-defaults" query parameter is set to "trim", all "in use" values are returned, except that the output is filtered to exclude any values that match the default defined in the YANG schema.

Servers are not required to support all values in the "with-defaults" query parameter on the operational state datastore. If a request is made using a value that is not supported, then the error handling behavior is as described in ([RFC8040], Section 4.8.9).

3.2.2. New "with-origin" Query Parameter

A new query parameter named "with-origin" is added to the GET operation. If present, it requests that the server includes "origin" metadata annotations in its response, as detailed in the NMDA. This parameter is only valid when querying {+restconf}/ds/ietf-datastores:operational or any datastores with identities derived from the "operational" identity. Otherwise, if an invalid datastore is specified then the server MUST return a response with a "400 Bad Request" status-line, using an error-tag value of "invalid-value". "origin" metadata annotations are not included unless a client explicitly requests them.
Data in the operational state datastore can come from multiple sources. The server should return the most accurate value for the "origin" metadata annotation as possible, indicating the source of the operational value, as specified in Section 5.3.4 of [RFC8342].

When encoding the origin metadata annotation for a hierarchy of returned nodes, the annotation can be omitted for a child node when the value matches that of the parent node, as described in "ietf-origin" YANG module [RFC8342].

The "with-origin" query parameter is optional to support. It is identified with the URI:

```
urn:ietf:params:restconf:capability:with-origin:1.0
```

4. IANA Considerations

This document defines two capability identifier URNs in the "RESTCONF Capability URNs" registry defined in [RFC8040]:

<table>
<thead>
<tr>
<th>Index</th>
<th>Capability Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>:with-origin</td>
<td>urn:ietf:params:restconf:capability:with-origin:1.0</td>
</tr>
<tr>
<td>:with-operational-defaults</td>
<td>urn:ietf:params:restconf:capability:with-operational-defaults:1.0</td>
</tr>
</tbody>
</table>

5. Security Considerations

This documents extends the RESTCONF protocol by introducing new datastore resources. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246]. The RESTCONF protocol uses the network configuration access control model [RFC8341], which provides the means to restrict access for particular RESTCONF users to a preconfigured subset of all available RESTCONF protocol operations and content.

The security constraints for the base RESTCONF protocol (see Section 12 of [RFC8040]) apply to the new RESTCONF datastore resources defined in this document.
6. Normative References

[I-D.ietf-netconf-rfc7895bis]
Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K.,
and R. Wilton, "YANG Library", draft-ietf-netconf-
rfc7895bis-06 (work in progress), April 2018.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/
rfc2119>.

rfc5246>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC
7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-
editor.org/info/rfc7951>.

Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017,

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC
2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,

Access Control Model", STD 91, RFC 8341, DOI 10.17487/

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K.,
and R. Wilton, "Network Management Datastore Architecture
(NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018,

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Abstract

This document defines a new notification message format, using yang-data. Included are:

- a new notification mechanism and encoding to replace the one way operation of RFC-5277
- a set of common, transport agnostic message header objects.
- how to bundle multiple event records into a single notification message.
- how to ensure these new capabilities are only used with capable receivers.

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 August 24, 2018.
1. Introduction

Mechanisms to support subscription to event notifications and yang datastore push are being defined in [I-D.draft-ietf-netconf-subscribed-notifications] and [I-D.ietf-netconf-yang-push]. Work on those documents has shown that notifications described in [RFC7950] section 7.16 could benefit from transport independent headers. Communicating the following
information to receiving applications can be done without explicit linkage to an underlying transport protocol:

- the time information was generated
- the time the information was placed in a message and queued for transport
- a signature to verify authenticity
- the process generating the information
- an originating request correlation
- an ability to bundle information records into one a message
- the ability to check for message loss/reordering

The document describes information elements needed for the functions above. It also provides YANG structures for sending messages containing one or more events and/or update records to a receiver.

2. 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The definition of notification is in RFC 7950 [RFC7950]. Publisher, receiver, and subscription are defined in [I-D.draft-ietf-netconf-subscribed-notifications].

3. Header Objects

There are a number of transport independent headers which should have common definition. These include:

- subscription-id: provides a reference into the reason the publisher believed the receiver wishes to be notified of this specific information.
- notification-time: the time an event, datastore update, or other item is recognized and recorded within the publisher.
- notification-id: Identifies the name of the notification, per the YANG notification statement. May also provide the name of a yang-
data statement (whether transporting other types of messages is in
scope is tbd).

- observation-domain-id: identifies the publisher process which
discovered and recorded the event notification. (note: look to
reuse the domains set up with IPFIX.)

- message-time: the time the message was packaged sent to the
transport layer for delivery to the receiver.

- signature: allows an application to sign a message so that a
receiver can verify the authenticity of the message.

- message-id: for a specific message generator, this identifies a
message which includes one or more event records. The message-id
increments by one with sequential messages.

- message-generator-id: identifier for the process which created the
message. This allows disambiguation of an information source,
such as the identification of different line cards sending the
messages. Used in conjunction with previous-message-id, this can
help find drops and duplications when messages are coming from
multiple sources on a device. If there is a message-generator-id
in the header, then the previous-message-id MUST be the message-id
from the last time that message-generator-id was sent.

4. Encapsulation of Header Objects in Messages

A specific set of well-known objects are of potential use to
networking layers prior being interpreted by some receiving
application layer process. By exposing this object information as
part of a header, and by using standardized object names, it becomes
possible for this object information to be leveraged in transit.

The objects defined in the previous section are these well-known
header objects. These objects are identified within a dedicated
header subtree which leads off a particular transportable message.
This allows header objects to be easily be decoupled, stripped, and
processed separately.

There are two types of transportable messages: one format is used
when there is one notification being encapsulated, and another format
used when there are many notifications being bundled into one
message.

A receiver which supporting this document MUST be able to handle
receipt of either type of message from an publisher. It is possible
that changes between message types can occur without any prior indication.

4.1. One Notification per Message

This section will be re-instated if NETCONF WG members are not comfortable with the efficiency of the solution which can encode many notifications per message described below.

4.2. Many Notifications per Message

While possible in some scenarios, it often inefficient to marshal and transport every notification independently. Instead, scale and processing speed can be improved by placing multiple notifications into one transportable bundle.

The format of this bundle appears in the yata-data tree below, and is more completely defined in the yang module. There are three parts of this bundle:

- A message header describing the marshaling, including information such as when the marshaling occurred
- A list of encapsulated information
- An optional message footer for whole-message signing and message-generator integrity verification.

Within the list of encapsulated notifications, there are also three parts:

- A notification header defining what is in an encapsulated notification
- The actual notification itself
- An optional notification footer for individual notification signing and observation-domain integrity verification.
yang-data message
  +--ro message!
    +--ro message-header
      |   +--ro message-time yang:date-and-time
      |   +--ro message-id? uint32
      |   +--ro message-generator-id? string
      |   +--ro notification-count? uint16
    +--ro notifications*
      +--ro notification-header
        |   +--ro notification-time yang:date-and-time
        |   +--ro yang-module? yang:yang-identifier
        |   +--ro yang-notification-name? notification-type
        |   +--ro subscription-id* uint32
        |   +--ro notification-id? uint32
        |   +--ro observation-domain-id? string
        +--ro notification-contents?
        +--ro notification-footer!
          +--ro signature-algorithm string
          +--ro signature-value string
          +--ro integrity-evidence? string
    +--ro message-footer!
      +--ro signature-algorithm string
      +--ro signature-value string
      +--ro integrity-evidence? string

An XML instance of a message might look like:
<yang-data bundled-message xmlns="urn:ietf:params:xml:ns:yang:ietf-notification-messages:1.0">
  <message-header>
    <message-time>2017-02-14T00:00:05Z</message-time>
    <message-id>456</message-id>
    <notification-count>2</notification-count>
  </message-header>
  <notifications>
    <notification>
      <notification-header>
        <notification-time>2017-02-14T00:00:02Z</notification-time>
        <subscription-id>823472</subscription-id>
        <yang-module>ietf-yang-push</yang-module>
        <yang-notification-name>push-change-update</yang-notification-name>
      </notification-header>
      <notification-contents>
          <datastore-changes-xml>
            <alpha xmlns="http://example.com/sample-data/1.0">
              <beta urn="urn:ietf:params:xml:ns:netconf:base:1.0:operation="delete"/>
            </alpha>
          </datastore-changes-xml>
        </push-change-update>
      </notification-contents>
    </notification>
    ...(record #2)...
  </notifications>
</yang-data>
5. Configuration of Headers

A publisher MUST select the set of headers to use within any particular message. The two mandatory headers which MUST always be applied are ‘message-time’ and ‘subscription-id’

Beyond these two mandatory headers, additional headers MAY be included. Configuration of what these optional headers should be can come from the following sources:

1. Publisher wide default headers for all notifications. These are included if an optional header is inserted into ‘additional-headers’ leaf-list shown in the yang tree below.

2. More notification specific headers may also be desired. If new headers are needed for a specific type of YANG notification, these can be populated through ‘additional-notification-headers’ leaf-list.

3. An application process may also identify common headers to use when transporting notifications for a specific subscription. How these are identified to a publisher is out-of-scope.

The set of headers used for any particular message is the superset of headers for the items listed above.

The YANG tree showing elements of configuration is depicted in the following figure.

module: ietf-notification-messages
    +--rw additional-default-headers {publisher}? optional-header
        +--rw additional-headers* optional-header
            +--rw yang-notification-specific-default* [yang-module yang-notification-name]
                +--rw yang-module yang:yang-identifier
                +--rw yang-notification-name notification-type
                +--rw additional-notification-headers* optional-notification-header

Configuration Model structure

Of note in this tree is the optional feature of ‘publisher’. This feature indicates an ability to send notifications. A publisher supporting this specification MUST also be able to parse any messages received as defined in this document.
6. Discovering Receiver Support

We need capability exchange from the receiver to the publisher at
transport session initiation to indicate support for this
specification.

For all types of transport connections, if the receiver indicates
support for this specification, then it MAY be used. In addition,
[RFC5277] one-way notifications MUST NOT be used if the receiver
indicates support for this specification to a publisher which also
supports it.

Where NETCONF transport is used, advertising this specification’s
namespace during an earlier client capabilities discovery phase MAY
be used to indicate support for this specification:

```xml
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>
      urn:ietf:params:xml:ns:yang:ietf-notification-messages:1.0
    </capability>
  </capabilities>
  <session-id>4</session-id>
</hello>
```

NOTE: It is understood that even though it is allowed in [RFC6241]
section 8.1, robust NETCONF client driven capabilities exchange is
not something which is common in implementation. Therefore reviewers
are asked to submit alternative proposals to the mailing list.

For RESTCONF, a mechanism for capability discovery is TBD. Proposals
are also welcome here.

The mechanism described above assumes that a capability discovery
phase happens before a subscription is started. This is not always
the case. As an example, consider HTTP2 configured subscriptions
from section 3.1.3 of [I-D.draft-ietf-netconf-restconf-notif], there
is no guarantee that a capability exchange has taken place before the
updates are pushed. A solution for this could be that a receiver
would reply "ok" and reply with the client capabilities as part of
the POST. (Or just use a different HTTP status code like 202 instead
of 200 'ok'). As such a requirement creates a new dependency for
[I-D.draft-ietf-netconf-restconf-notif] upon this specification, more
discussion is required to decide if this is a viable solution.
7. YANG Module

<CODE BEGINS> file "ietf-notification-messages@2018-01-31.yang"

module ietf-notification-messages {  
  yang-version 1.1;  
  namespace  
    "urn:ietf:params:xml:ns:yang:ietf-notification-messages";  
  prefix nm;  

  import ietf-yang-types { prefix yang; }  
  import ietf-restconf { prefix rc; }

  organization "IETF";
  contact  
    "WG Web: <http://tools.ietf.org/wg/netconf/>  
    WG List: <mailto:netconf@ietf.org>  
    Editor: Eric Voit  
      <mailto:evoit@cisco.com>  
    Editor: Henk Birkholz  
      <mailto:henk.birkholz@sit.fraunhofer.de>  
    Editor: Alexander Clemm  
      <mailto:ludwig@clemm.org>  
    Editor: Andy Bierman  
      <mailto:andy@yumaworks.com>  
    Editor: Tim Jenkins  
      <mailto:timjenki@cisco.com>";

  description  
    "This module contains conceptual YANG specifications for yang-data  
    messages carrying notifications with well known header objects.";

  revision 2018-01-31 {  
    description  
      "Initial version.";

    reference  
      "draft-ietf-netconf-notification-messages-03";
  }

  /*
  * FEATURES

feature publisher {
    description
        "This feature indicates that support for both publisher and receiver of messages complying to the specification.";
}

/* IDENTITIES */

/* Identities for common headers */

identity common-header {
    description
        "A well known header which can be included somewhere within a message.";
}

identity message-time {
    base common-header;
    description
        "Header information consisting of time the message headers were placed generated prior to being sent to transport";
}

identity subscription-id {
    base common-header;
    description
        "Header information consisting of the identifier of the subscription associated with the notification being encapsulated.";
}

identity notification-count {
    base common-header;
    description
        "Header information consisting of the quantity of notifications in a bundled-message for a specific receiver.";
}

identity optional-header {
    base common-header;
    description
        "A well known header which an application may choose to include within a message.";
}

identity message-id {
    base optional-header;
    description
     "Header information that identifies a message to a specific
     receiver";
}

identity message-generator-id {
    base optional-header;
    description
     "Header information consisting of an identifier for a software
     entity which created the message (e.g., linecard 1).";
}

identity message-signature {
    base optional-header;
    description
     "Identifies two elements of header information consisting of a
     signature and the signatature type for the contents of a message.
     Signatures can be useful for originating applications to
     verify record contents even when shipping over unsecure
     transport.";
}

identity message-integrity-evidence {
    base optional-header;
    description
     "Header information consisting of the information which backs up
     the assertions made as to the validity of the information
     provided within the message.";
}

identity optional-notification-header {
    base optional-header;
    description
     "A well known header which an application may choose to include
     within a message.";
}

identity notification-time {
    base optional-notification-header;
    description
     "Header information consisting of the time an originating process
     created the notification.";
}

identity notification-id {

base optional-notification-header;
description
    "Header information consisting of an identifier for an instance
    of a notification egressing a publisher. ";
}

identity observation-domain-id {
    base optional-notification-header;
description
    "Header information identifying the software entity which created
    the notification (e.g., process id).";
}

identity notification-signature {
    base optional-notification-header;
description
    "Header information consisting of the information which backs up
    the assertions made as to the validity of the information
    provided within the notification.";
}

identity notification-integrity-evidence {
    base optional-notification-header;
description
    "Header information consisting of the information which backs up
    the assertions made as to the validity of the information
    provided within the notification.";
}

/
*  * TYPEDEFs
  */

typedef optional-header {
    type identityref {
        base optional-header;
    }
description
    "Type of header object which may be included somewhere within a
    message.";
}

typedef optional-notification-header {
    type identityref {
        base optional-notification-header;
    }

typedef notification-type {
  type string {
    pattern '[a-zA-Z_]\[a-zA-Z0-9\-_\.]*';
  }
  description
    "The name of a notification within a YANG module.";
  reference
    "RFC-7950 Section 7.16";
}
/*
 * GROUPINGS
 */
grouping message-header {
  description
    "Header information included with a message.";
  leaf message-time {
    type yang:date-and-time;
    mandatory true;
    description
      "time the message was generated prior to being sent to
       transport.";
  }
  leaf message-id {
    type uint32;
    description
      "Id for a message going to a receiver from a message
generator. The id will increment by one with each message sent
from a particular message generator, allowing the message-id
to be used as a sequence number.";
  }
  leaf message-generator-id {
    type string;
    description
      "Software entity which created the message (e.g., linecard 1).
The combination of message-id and message-generator-id must be
unique until reset or a roll-over occurs.";
  }
  leaf notification-count {
    type uint16;
    description
      "Quantity of notification records in a bundled-message
specific receiver.
}
}

grouping notification-within-a-module {
  description
  "A location of a notification within a yang model."
  leaf yang-module {
    type yang:yang-identifier;
    description
    "Name of the YANG module supported by the publisher."
  }
  leaf yang-notification-name {
    type notification-type;
    description
    "The name of a notification likely from a YANG module. Note that this object should be in the notification contents, so a debate is needed whether this is redundant."
  }
}

grouping notification-header {
  description
  "Common informational objects which might help a receiver interpret the meaning, details, or importance of a notification."
  leaf notification-time {
    type yang:date-and-time;
    mandatory true;
    description
    "Time the system recognized the occurrence of an event."
  }
  uses notification-within-a-module;
  leaf-list subscription-id {
    type uint32;
    description
    "Id of the subscription which led to the notification being generated."
  }
  leaf notification-id {
    type uint32;
    description
    "Identifier for the notification record."
  }
  leaf observation-domain-id {
    type string;
    description
    "Software entity which created the notification record (e.g., process id)."
  }
}
grouping security-footer {
  description
    "Reusable grouping for common objects which apply to the the
    signing of notifications or messages.";
  leaf signature-algorithm {
    type string;
    mandatory true;
    description
    "The technology with which an originator signed of some
delineated contents.";
  }
  leaf signature-value {
    type string;
    mandatory true;
    description
    "Any originator signing of the contents of a header and
content. This is useful for verifying contents even when
shipping over unsecure transport.";
  }
  leaf integrity-evidence {
    type string;
    description
    "This mechanism allows a verifier to ensure that the use of the
private key, represented by the corresponding public key
certificate, was performed with a TCG compliant TPM
environment. This evidence is never included in within any
signature.";
    reference
    "TCG Infrastructure Workgroup, Subject Key Attestation Evidence
Extension, Specification Version 1.0, Revision 7.";
  }
}

/*
 * YANG-DATA messages for receivers
 */

rc:yang-data message {
  container message {
    presence
    "Indicates attempt to communicate notifications to a receiver.";
    description
    "Message to a receiver containing one or more notifications";
}
container message-header {
  description
    "Header info for messages.";
  uses message-header;
}
list notifications {
  description
    "Set of notifications to a receiver.";
  container notification-header {
    description
      "Header info for a notification.";
    uses notification-header;
  }
  anydata notification-contents {
    description
      "Encapsulates objects following YANG’s notification-stmt grammar of RFC-7950 section 14. Within are the notified objects the publisher actually generated in order to be passed to a receiver after all filtering has completed.";
  }
  container notification-footer {
    presence
      "Indicates attempt to secure a notification.";
    description
      "Signature and evidence for messages.";
    uses security-footer;
  }
}
container message-footer {
  presence
    "Indicates attempt to secure the entire message.";
  description
    "Signature and evidence for messages.";
  uses security-footer;
}

/*
 * DATA-NODES
 */

container additional-default-headers {
  if-feature "publisher";
  description
    "This container maintains a list of which additional notifications should use which optional headers if the receiver supports this specification.";
}
leaf-list additional-headers {
  type optional-header;
  description
       "This list contains the identities of the optional header types
        which are to be included within each message from this
        publisher.";
}
list yang-notification-specific-default {
  key "yang-module yang-notification-name";
  description
       "For any included YANG notifications, this list provides
        additional optional headers which should be placed within the
        container notification-header if the receiver supports this
        specification. This list incrementally adds to any headers
        indicated within the leaf-list 'additional-headers'.";
  uses notification-within-a-module;
  leaf-list additional-notification-headers {
    type optional-notification-header;
    description
       "The set of additional default headers which will be sent
        for a specific YANG notification.";
  }
}

8. Backwards Compatibility

With this specification, there is no change to YANG's 'notification'
statement

Legacy clients are unaffected, and existing users of [RFC5277],
[RFC7950], and [RFC8040] are free to use current behaviors until all
involved device support this specification.

9. Security Considerations

Certain headers might be computationally complex for a publisher to
deliver. Signatures or encryption are two examples of this. It MUST
be possible to suspend or terminate a subscription due to lack of
resources based on this reason.

Decisions on whether to bundle or not to a receiver are fully under
the purview of the Publisher. A receiver could slow delivery to
existing subscriptions by creating new ones. (Which would result in
the publisher going into a bundling mode.)
10. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Martin Bjorklund, Einar Nilsen-Nygard, and Kent Watsen.

11. References

11.1. Normative References

[I-D.draft-ietf-netconf-subscribed-notifications]


11.2. Informative References

[I-D.draft-ietf-netconf-restconf-notif]

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

Appendix A. Changes between revisions

(To be removed by RFC editor prior to publication)

v02 - v03

- Removed the option for an unbundled message. This might be re-added later for transport efficiency if desired by the WG
- New message structure driven by the desire to put the signature information at the end.

v01 - v02

- Fixed the yang-data encapsulation container issue
- Updated object definitions to point to RFC-7950 definitions
- Added headers for module and notification-type.

v00 - v01

- Alternative to 5277 one-way notification added
- Storage of default headers by notification type
- Backwards compatibility
- Capability discovery
- Move to yang-data
- Removed dscp and record-type as common headers. (Record type can be determined by the namespace of the record contents. Dscp is useful where applications need internal communications within a Publisher, but it is unclear as to whether this use case need be exposed to a receiver.)
Appendix B. Issues being worked

(To be removed by RFC editor prior to publication)

Is this capability just for notifications, or is it for any yang-data element too?

A complete JSON document is supposed to be sent as part of Media Type "application/yang-data+json". As we are sending separate notifications after each other, we need to choose whether we start with some extra encapsulation for the very first message pushed, or if we want a new Media Type for streaming updates.

Improved discovery mechanisms for NETCONF

Should we defer support for HTTP2 configured subscriptions until this draft is available? Without capabilities exchange, it might just be easier to wait. In addition, JSON encoding still needs a notification type which is not existing or represented in referenceable in existing yang-models.

Need to ensure the proper references exist to a notification definition driven by RFC-7950 which is acceptable to other eventual users of this specification.

We need to link to Andy Bierman’s anydata extensibility draft for informational purposes. This is under a WG adoption call.

Appendix C. Subscription Specific Headers

(To be removed by RFC editor prior to publication)

This section discusses a future functional addition which could leverage this draft. It is included for informational purposes only.

A dynamic subscriber might want to mandate that certain headers be used for push updates from a publisher. Some examples of this include a subscriber requesting to:

- establish this subscription, but just if transport messages containing the pushed data will be encrypted,
- establish this subscription, but only if you can attest to the information being delivered in requested notification records, or
- provide a sequence-id for all messages to this receiver (in order to check for loss).
Providing this type of functionality would necessitate a new revision of the [I-D.draft-ietf-netconf-subscribed-notifications]’s RPCs and state change notifications. Subscription specific header information would overwrite the default headers identified in this document.

Appendix D. Implications to Existing RFCs

(To be removed by RFC editor prior to publication)

YANG one-way exchanges currently use a non-extensible header and encoding defined in section 4 of RFC-5277. These RFCs MUST be updated to enable this draft. These RFCs SHOULD be updated to provide examples

D.1. Implications to RFC-7950

Sections which expose netconf:capability:notification:1.0 are 4.2.10

Sections which provide examples using netconf:notification:1.0 are 7.10.4, 7.16.3, and 9.9.6

D.2. Implications to RFC-8040

Section 6.4 demands use of RFC-5277’s netconf:notification:1.0, and later in the section provides an example.

Authors’ Addresses

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Henk Birkholz
Fraunhofer SIT

Email: henk.birkholz@sit.fraunhofer.de

Andy Bierman
YumaWorks

Email: andy@yumaworks.com
Alexander Clemm
Huawei
Email: ludwig@clemm.org

Tim Jenkins
Cisco Systems
Email: timjenki@cisco.com
Abstract

This document defines two YANG modules, one module to configure a RESTCONF client and the other module to configure a RESTCONF server. Both modules support the TLS transport protocol with both standard RESTCONF and RESTCONF Call Home connections.

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.

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

- I-D.ietf-netconf-keystore
- I-D.ietf-netconf-tls-client-server

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

- "XXXX" --> the assigned RFC value for this draft
- "ZZZZ" --> the assigned RFC value for I-D.ietf-netconf-tls-client-server

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

- "2018-06-04" --> the publication date of this draft

The following Appendix section is to be removed prior to publication:
o Appendix A. 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 December 6, 2018.

Copyright Notice

Copyright (c) 2018 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 ............................................ 3
   1.1. Terminology ..................................... 3
2. The RESTCONF Client Model ................................ 3
   2.1. Tree Diagram .................................. 4
   2.2. Example Usage .................................. 6
   2.3. YANG Module .................................. 8
3. The RESTCONF Server Model ............................... 17
   3.1. Tree Diagram .................................. 17
   3.2. Example Usage .................................. 20
   3.3. YANG Module .................................. 23
4. Security Considerations ............................... 32
5. IANA Considerations ................................. 33
1. Introduction

This document defines two YANG [RFC7950] modules, one module to configure a RESTCONF client and the other module to configure a RESTCONF server [RFC8040]. Both modules support the TLS [RFC5246] transport protocol with both standard RESTCONF and RESTCONF Call Home connections [RFC8071].

1.1. 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. The RESTCONF Client Model

The RESTCONF client model presented in this section supports both clients initiating connections to servers, as well as clients listening for connections from servers calling home.

This model, like that presented in [I-D.ietf-netconf-netconf-client-server], is designed to support any number of possible transports. RESTCONF only supports the TLS transport currently, thus this model only supports the TLS transport.

All private keys and trusted certificates are held in the keystore model defined in [I-D.ietf-netconf-keystore].

YANG feature statements are used to enable implementations to advertise which parts of the model the RESTCONF client supports.
2.1. Tree Diagram

The following tree diagram [RFC8340] provides an overview of the data model for the "ietf-restconf-client" module. Just the container is displayed below, but there is also a reusable grouping by the same name that the container is using.

[Note: ‘\’ line wrapping for formatting only]

```text
module: ietf-restconf-client
  +--rw restconf-client
    +--rw initiate! {initiate}?
      +--rw restconf-server* [name]
        +--rw name string
        +--rw endpoints
          +--rw endpoint* [name]
            +--rw name string
            +--rw (transport)
              +--:(tls) {tls-initiate}?
                +--rw tls
                  +--rw address inet:host
                  +--rw port? inet:port-number
                +--:(certificate)
                  +--rw (local-or-keystore)
                    +--:(local)
                    +--rw algorithm
                      +--rw public-key binary
                      +--rw private-key union
                      +--rw cert ct:end-entity-ce
                        +--n certificate-expiration
                          +-- expiration-date? yang:date-and-time
                          +--:(keystore)
                            {keystore-implemented}
                +--rw reference
```

Watsen & Wu             Expires December 6, 2018                [Page 4]
y-certificate-ref
  +--rw server-auth
    |       +--rw pinned-ca-certs?
    |          |       ta:pinned-certificates-ref
    |       +--rw pinned-server-certs?
    |          ta:pinned-certificates-ref
    +--rw hello-params
      |       (tls-client-hello-params-config)?
      |           +--rw tls-versions
      |               +--rw tls-version* identityref
      |           +--rw cipher-suites
      |               +--rw cipher-suite* identityref
    +--rw connection-type
      |       (connection-type)?
      |           +--:(persistent-connection)
      |               +--rw persistent!
      |               |       +--rw idle-timeout? uint32
      |               |       +--rw keep-aliases
      |               |       +--rw max-wait? uint16
      |               |       +--rw max-attempts? uint8
      |           +--:(periodic-connection)
      |               +--rw periodic!
      |               |       +--rw idle-timeout? uint16
      |               +--rw reconnect-timeout? uint16
    +--rw reconnect-strategy
      |       start-with? enumeration
      +--rw max-attempts? uint8
    +--rw listen! (listen)?
      |       idle-timeout? uint16
    +--rw endpoint* [name]
      |       name string
      +--rw (transport) (tls-listen)?
        +--:(tls) (tls-listen)?
          |       address? inet:ip-address
          |       port? inet:port-number
      +--rw client-identity
        |       (auth-type)
          +--:(certificate)
            |       +--rw certificate
            |           +--rw (local-or-keystore)
            |               +--:(local)
            |               |       algorithm
            |               |          ct:algorithm-ref
            |               |       +--rw public-key
            |               |          binary
            |               +--rw private-key
2.2. Example Usage

The following example illustrates configuring a RESTCONF client to initiate connections, as well as listening for call-home connections.

This example is consistent with the examples presented in Section 2.2 of [I-D.ietf-netconf-keystore].

[Note: ‘\’ line wrapping for formatting only]

```xml
<restconf-client
    xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-client">
  <!-- RESTCONF servers to initiate connections to -->
  <initiate>
    <restconf-server>
      <name>corp-fw1</name>
      <endpoints>
        <endpoint>
          <name>corp-fw1.example.com</name>
          <tls>
            <address>corp-fw1.example.com</address>
            <client-identity>
              ++-rw server-auth
              |  +%rw pinned-ca-certs?
              |  |   ta:pinned-certificates-ref
              |  +%rw pinned-server-certs?
              |   ta:pinned-certificates-ref
              ++-rw hello-params
              |    {tls-client-hello-params-config}?
              |      ++-rw tls-versions
              |      |        ++-rw tls-version* identityref
              |      ++-rw cipher-suites
              |      |        ++-rw cipher-suite* identityref
```
<certificate>
  <private-key>base64encodedvalue==</private-key>
  <public-key>base64encodedvalue==</public-key>
  <cert>base64encodedvalue==</cert>
</certificate>

</client-identity>
</server-auth>
<pinned-ca-certs>explicitly-trusted-server-ca-certs</pinned-ca-certs>
<pinned-server-certs>explicitly-trusted-server-certs</pinned-server-certs>
</server-auth>
</endpoint>
</endpoints>
</restconf-server>
</initiate>

<!-- endpoints to listen for RESTCONF Call Home connections on -->

<listen>
  <endpoint>
    <name>Intranet-facing listener</name>
    <tls>
      <address>11.22.33.44</address>
    </tls>
  </endpoint>
</listen>
<client-identity>
    <certificate>
        <private-key>base64encodedvalue==</private-key>
        <public-key>base64encodedvalue==</public-key>
        <cert>base64encodedvalue==</cert>
    </certificate>
</client-identity>

<server-auth>
    <pinned-ca-certs>explicitly-trusted-server-ca-certs</pinned-ca-certs>
    <pinned-server-certs>explicitly-trusted-server-certs</pinned-server-certs>
</server-auth>

</endpoint>
</listen>
</restconf-client>

2.3. YANG Module

This YANG module has normative references to [RFC6991], [RFC8040], and [RFC8071], and [I-D.ietf-netconf-tls-client-server].

<CODE BEGINS> file "ietf-restconf-client@2018-06-04.yang"

<CODE BEGINS>

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

  prefix "rcc";

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-tls-client {
    prefix ts;
    revision-date 2018-06-04; // stable grouping definitions
    reference
      "RFC ZZZZ: YANG Groupings for TLS Clients and TLS Servers";
  }

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

<CODE ENDS>

Watsen & Wu             Expires December 6, 2018                [Page 8]
description
"This module contains a collection of YANG definitions for configuring RESTCONF clients.

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

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

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

revision "2018-06-04" {
  description
    "Initial version";
  reference
    "RFC XXXX: RESTCONF Client and Server Models";
}

// Features

feature initiate {
  description
    "The 'initiate' feature indicates that the RESTCONF client supports initiating RESTCONF connections to RESTCONF servers using at least one transport (e.g., TLS, etc.).";
}

feature tls-initiate {
  if-feature initiate;
  description
    "The 'tls-initiate' feature indicates that the RESTCONF client...";
supports initiating TLS connections to RESTCONF servers. This feature exists as TLS might not be a mandatory to implement transport in the future.

reference
"RFC 8040: RESTCONF Protocol";

} feature listen {
    description
    "The 'listen' feature indicates that the RESTCONF client supports opening a port to accept RESTCONF server call home connections using at least one transport (e.g., TLS, etc.).";

} feature tls-listen {
    if-feature listen;
    description
    "The 'tls-listen' feature indicates that the RESTCONF client supports opening a port to listen for incoming RESTCONF server call-home TLS connections. This feature exists as TLS might not be a mandatory to implement transport in the future.";

    reference
    "RFC 8071: NETCONF Call Home and RESTCONF Call Home";

} container restconf-client {
    uses restconf-client;
    description
    "Top-level container for RESTCONF client configuration.";

} grouping restconf-client {
    description
    "Top-level grouping for RESTCONF client configuration.";

} container initiate {
    if-feature initiate;
    presence "Enables client to initiate TCP connections";
    description
    "Configures client initiating underlying TCP connections.";

    list restconf-server {
        key name;
        min-elements 1;
        description
        "List of RESTCONF servers the RESTCONF client is to initiate connections to in parallel.";
    }


leaf name {
  type string;
  description
  "An arbitrary name for the RESTCONF server.";
}

container endpoints {
  description
  "Container for the list of endpoints.";
  list endpoint {
    key name;
    min-elements 1;
    ordered-by user;
    description
    "A non-empty user-ordered list of endpoints for this
    RESTCONF client to try to connect to in sequence.
    Defining more than one enables high-availability.";
    leaf name {
      type string;
      description
      "An arbitrary name for this endpoint.";
    }
  }
}

choice transport {
  mandatory true;
  description
  "Selects between available transports. This is a
  'choice' statement so as to support additional
  transport options to be augmented in.";
  case tls {
    if-feature tls-initiate;
    container tls {
      description
      "Specifies TLS-specific transport
      configuration.";
      leaf address {
        type inet:host;
        mandatory true;
        description
        "The IP address or hostname of the endpoint.
        If a domain name is configured, then the
        DNS resolution should happen on each usage
        attempt. If the the DNS resolution results
        in multiple IP addresses, the IP addresses
        will be tried according to local preference
        order until a connection has been established
        or until all IP addresses have failed.";
      }
      leaf port {
        type inet:port-number;
      }
    }
  }
}
default 443;
description
"The IP port for this endpoint. The RESTCONF client will use the IANA-assigned well-known port for 'https' (443) if no value is specified."
}
uses ts:tls-client-grouping {
refine "client-identity/auth-type" {
  mandatory true;
description
  "RESTCONF clients MUST pass some authentication credentials.";
}
}
} // end tls
} // end transport
container connection-type {
description
  "Indicates the kind of connection to use."
choice connection-type {
default persistent-connection;
description
  "Selects between available connection types."
case persistent-connection {
container persistent {
presence
  "Indicates that a persistent connection is to be maintained."
description
  "Maintain a persistent connection to the RESTCONF server. If the connection goes down, immediately start trying to reconnect to it, using the reconnection strategy. This connection type minimizes any RESTCONF server to RESTCONF client data-transfer delay, albeit at the expense of holding resources longer."
leaf idle-timeout {
type uint32;
units "seconds";
default 86400; // one day;
description
  "Specifies the maximum number of seconds that the underlying TLS session may remain idle. A TLS session will be dropped if it is idle for an interval longer than this number of seconds. If set to zero, then
the client will never drop a session because it is idle. Sessions that have a notification subscription active are never dropped."

}

container keep-alives {
    description
    "Configures the keep-alive policy, to proactively test the aliveness of the TLS server. An unresponsive TLS server will be dropped after approximately max-attempts * max-wait seconds.";
    reference
    "RFC 8071: NETCONF Call Home and RESTCONF Call Home, Section 3.1, item S6";
    leaf max-wait {
        type uint16 {
            range "1..max";
        }
        units seconds;
        default 30;
        description
        "Sets the amount of time in seconds after which if no data has been received from the TLS server, a TLS-level message will be sent to test the aliveness of the TLS server.";
    }
    leaf max-attempts {
        type uint8;
        default 3;
        description
        "Sets the maximum number of sequential keep-alive messages that can fail to obtain a response from the TLS server before assuming the TLS server is no longer alive.";
    }
}

}

}

}
so that, e.g., the RESTCONF client can collect data (logs) from the RESTCONF server. Once the connection is closed, for whatever reason, the RESTCONF client will restart its timer until the next connection.

leaf idle-timeout {
  type uint16;
  units "seconds";
  default 300; // five minutes
  description
  "Specifies the maximum number of seconds that the underlying TLS session may remain idle. A TLS session will be dropped if it is idle for an interval longer than this number of seconds. If set to zero, then the server will never drop a session because it is idle.";
}

leaf reconnect-timeout {
  type uint16 {
    range "1..max";
  }
  units minutes;
  default 60;
  description
  "Sets the maximum amount of unconnected time the RESTCONF client will wait before re-establishing a connection to the RESTCONF server. The RESTCONF client may initiate a connection before this time if desired (e.g., to set configuration).";
}

} // end periodic-connection
} // end connection-type
} // end connection-type
container reconnect-strategy {
  description
  "The reconnection strategy directs how a RESTCONF client reconnects to a RESTCONF server, after discovering its connection to the server has dropped, even if due to a reboot. The RESTCONF client starts with the specified endpoint and tries to connect to it max-attempts times before trying the next endpoint in the list (round robin).";

  leaf start-with {
    type enumeration {

enum first-listed {
    description
    "Indicates that reconnections should start
    with the first endpoint listed.";
}

enum last-connected {
    description
    "Indicates that reconnections should start
    with the endpoint last connected to. If
    no previous connection has ever been
    established, then the first endpoint
    configured is used. RESTCONF clients
    SHOULD be able to remember the last
    endpoint connected to across reboots.";
}

default first-listed;

description
    "Specifies which of the RESTCONF server’s
    endpoints the RESTCONF client should start
    with when trying to connect to the RESTCONF
    server.";

leaf max-attempts {
    type uint8 {
        range "1..max";
    }
    default 3;
    description
    "Specifies the number times the RESTCONF client
    tries to connect to a specific endpoint before
    moving on to the next endpoint in the list
    (round robin).";
}

} // end reconnect-strategy
} // end endpoint
} // end endpoints
} // end restconf-server
} // end initiate

container listen {
    if-feature listen;
    presence "Enables client to accept call-home connections";
    description
    "Configures client accepting call-home TCP connections.";

    leaf idle-timeout {
        type uint16;
units "seconds";
default 3600; // one hour
description
"Specifies the maximum number of seconds that an underlying TLS session may remain idle. A TLS session will be dropped if it is idle for an interval longer than this number of seconds. If set to zero, then the server will never drop a session because it is idle. Sessions that have a notification subscription active are never dropped.";
}

list endpoint {
key name;
min-elements 1;
description
"List of endpoints to listen for RESTCONF connections.";
leaf name {
type string;
description
"An arbitrary name for the RESTCONF listen endpoint.";
}
choice transport {
mandatory true;
description
"Selects between available transports. This is a 'choice' statement so as to support additional transport options to be augmented in.";
case tls {
if-feature tls-listen;
container tls {
description
"TLS-specific listening configuration for inbound connections.";
leaf address {
type inet:ip-address;
description
"The IP address to listen on for incoming call-home connections. The RESTCONF client will listen on all configured interfaces if no value is specified. INADDR_ANY (0.0.0.0) or INADDR6_ANY (0:0:0:0:0:0:0:0 a.k.a. ::) MUST be used when the server is to listen on all IPv4 or IPv6 addresses, respectively.";
}
leaf port {
type inet:port-number;
default 4336;
3. The RESTCONF Server Model

The RESTCONF server model presented in this section supports servers both listening for connections as well as initiating call-home connections.

All private keys and trusted certificates are held in the keystore model defined in [I-D.ietf-netconf-keystore].

YANG feature statements are used to enable implementations to advertise which parts of the model the RESTCONF server supports.

3.1. Tree Diagram

The following tree diagram [RFC8340] provides an overview of the data model for the "ietf-restconf-client" module. Just the container is displayed below, but there is also a reusable grouping by the same name that the container is using.

[Note: \ line wrapping for formatting only]

module: ietf-restconf-server
    +-rw restconf-server

Watsen & Wu             Expires December 6, 2018               [Page 17]
Internet-Draft      RESTCONF Client and Server Models          June 2018

++-rw listen! {listen}?
  +--rw endpoint* [name]
      +--rw name     string
      +--rw (transport)
          +--:(tls) {tls-listen}?
              +--rw tls
                  +--rw address? inet:ip-address
                  +--rw port?  inet:port-number
                  +--rw server-identity
                      +--rw (local-or-keystore)
                          +--:(local)
                              +--rw algorithm
                                  |  ct:key-algorithm-ref
                              +--rw public-key       binary
                              +--rw private-key      union
                              +--rw cert
                                  |  ct:end-entity-cert-cms
                                  +--n certificate-expiration
                                      +-- expiration-date?
                                          yang:date-and-time
                          +--:(keystore) {keystore-implemented}?
                              +--rw reference
                                  ks:asymmetric-key-certificate-ref
                              +--rw client-auth
                                  +--rw pinned-ca-certs?
                                      |  ta:pinned-certificates-ref
                                  +--rw pinned-client-certs?
                                      |  ta:pinned-certificates-ref
                                  +--rw cert-maps
                                      +--rw cert-to-name* [id]
                                          +--rw id     uint32
                                          +--rw fingerprint
                                              |  x509c2n:tls-fingerprint
                                          +--rw map-type       identityref
                                          +--rw name           string
                                  +--rw hello-params
                                      {tls-server-hello-params-config}?
                                          +--rw tls-versions
                                              |  +--rw tls-version* identityref
                                          +--rw cipher-suites
                                              +--rw cipher-suite* identityref
                                  +--rw call-home! {call-home}?
                                      +--rw restconf-client* [name]
                                          +--rw name     string
                                          +--rw endpoints
                                              +--rw endpoint* [name]
                                                  +--rw name     string

Watsen & Wu             Expires December 6, 2018               [Page 18]
+--rw (transport)
  +--:(tls) (tls-call-home)?
    +--rw tls
      +--rw address inet:host
      +--rw port? inet:port-number
    +--rw server-identity
      +--rw (local-or-keystore)
        +--:(local)
          +--rw algorithm
            |  ct:key-algorithm-ref
          +--rw public-key
            |  binary
          +--rw private-key
            |  union
          +--rw cert
            |  ct:end-entity-cert-cms
          +--n certificate-expiration
            +-- expiration-date?
              yang:date-and-time
        +--:(keystore) (keystore-implemented)?
          +--rw reference
            ks:asymmetric-key-certifi
cate-ref
          +--rw client-auth
            +--rw pinned-ca-certs?
              |  ta:pinned-certificates-ref
            +--rw pinned-client-certs?
              |  ta:pinned-certificates-ref
            +--rw cert-maps
              +--rw cert-to-name* [id]
                +--rw id uint32
                +--rw fingerprint
                  |  x509c2n:tls-fingerprint
                +--rw map-type identityref
                +--rw name string
          +--rw hello-params
            (tls-server-hello-params-config)?
            +--rw tls-versions
              |  +--rw tls-version* identityref
            +--rw cipher-suites
              |  +--rw cipher-suite* identityref
          +--rw connection-type
            +--rw (connection-type)?
              +--:(persistent-connection)
                +--rw persistent!
                  +--rw idle-timeout? uint32
                  +--rw keep-alives
3.2. Example Usage

The following example illustrates configuring a RESTCONF server to listen for RESTCONF client connections, as well as configuring call-home to one RESTCONF client.

This example is consistent with the examples presented in Section 2.2 of [I-D.ietf-netconf-keystore].

[Note: `\` line wrapping for formatting only]

```
<restconf-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-server"
  xmlns:x509c2n="urn:ietf:params:xml:ns:yang:ietf-x509-cert-to-name">
  <!-- endpoints to listen for RESTCONF connections on -->
  <listen>
    <endpoint>
      <name>netconf/tls</name>
      <tls>
        <address>11.22.33.44</address>
        <server-identity>
          <private-key>base64encodedvalue==</private-key>
          <public-key>base64encodedvalue==</public-key>
          <cert>base64encodedvalue==</cert>
        </server-identity>
        <client-auth>
          <pinned-ca-certs>explicitly-trusted-client-ca-certs</pinned-ca-certs>
          <pinned-client-certs>explicitly-trusted-client-certs</pinned-client-certs>
          <cert-maps>
            <cert-to-name>
              <id>1</id>
            </cert-to-name>
          </cert-maps>
        </client-auth>
      </tls>
    </endpoint>
  </listen>
</restconf-server>
```
<!-- call home to a RESTCONF client with two endpoints -->
<call-home>
 <restconf-client>
  <name>config-manager</name>
  <endpoints>
   <endpoint>
    <name>east-data-center</name>
    <tls>
     <address>22.33.44.55</address>
     <server-identity>
      <private-key>base64encodedvalue==</private-key>
      <public-key>base64encodedvalue==</public-key>
      <cert>base64encodedvalue==</cert>
     </server-identity>
     <client-auth>
      <pinned-ca-certs>explicitly-trusted-client-ca-certs</pinned-ca-certs>
      <pinned-client-certs>explicitly-trusted-client-certs</pinned-client-certs>
      <cert-maps>
       <cert-to-name>
        <id>1</id>
        <fingerprint>11:0A:05:11:00</fingerprint>
        <map-type>x509c2n:san-any</map-type>
       </cert-to-name>
       <cert-to-name>
        <id>2</id>
        <fingerprint>B3:4F:A1:8C:54</fingerprint>
        <map-type>x509c2n:specified</map-type>
        <name>scooby-doo</name>
       </cert-to-name>
      </cert-maps>
     </client-auth>
    </tls>
   </endpoint>
  </endpoints>
 </restconf-client>
</call-home>
Watsen & Wu             Expires December 6, 2018               [Page 22]
3.3. YANG Module

This YANG module has normative references to [RFC6991], [RFC7407], [RFC8040], [RFC8071], and [I-D.ietf-netconf-tls-client-server].

<CODE BEGINS> file "ietf-restconf-server@2018-06-04.yang"
module ietf-restconf-server {
  yang-version 1.1;

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

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-x509-cert-to-name {
    prefix x509c2n;
    reference
      "RFC 7407: A YANG Data Model for SNMP Configuration";
  }

  import ietf-tls-server {
    prefix ts;
    revision-date 2018-06-04; // stable grouping definitions
    reference
      "RFC ZZZZ: YANG Groupings for TLS Clients and TLS Servers";
  }

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

  contact
    "WG Web:  <http://datatracker.ietf.org/wg/netconf/>
    WG List:  <mailto:netconf@ietf.org>
    Author:  Kent Watsen
      <mailto:kwatsen@juniper.net>
    Author:  Gary Wu
      <mailto:garywu@cisco.com>

Watsen & Wu Expires December 6, 2018 [Page 23]
description
"This module contains a collection of YANG definitions for configuring RESTCONF servers.

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

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

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

revision "2018-06-04" {
  description
    "Initial version";
  reference
    "RFC XXXX: RESTCONF Client and Server Models";
}

// Features

feature listen {
  description
    "The 'listen' feature indicates that the RESTCONF server supports opening a port to accept RESTCONF client connections using at least one transport (e.g., TLS, etc.).";
}

feature tls-listen {
  if-feature listen;
  description
    "The 'tls-listen' feature indicates that the RESTCONF server supports opening a port to listen for incoming RESTCONF client connections. This feature exists as TLS might not be a mandatory to implement transport in the future.";
  reference
    "RFC 8040: RESTCONF Protocol";
}
feature call-home {
  description "The 'call-home' feature indicates that the RESTCONF server supports initiating RESTCONF call home connections to RESTCONF clients using at least one transport (e.g., TLS, etc.).";
  reference "RFC 8071: NETCONF Call Home and RESTCONF Call Home";
}

feature tls-call-home {
  if-feature call-home;
  description "The 'tls-call-home' feature indicates that the RESTCONF server supports initiating connections to RESTCONF clients. This feature exists as TLS might not be a mandatory to implement transport in the future.";
  reference "RFC 8071: NETCONF Call Home and RESTCONF Call Home";
}

container restconf-server {
  uses restconf-server;
  description "Top-level container for RESTCONF server configuration.";
}

grouping restconf-server {
  description "Top-level grouping for RESTCONF server configuration.";
}

container listen {
  if-feature listen;
  presence "Enables server to listen for TCP connections";
  description "Configures listen behavior";
  list endpoint {
    key name;
    min-elements 1;
    description "List of endpoints to listen for RESTCONF connections.";
    leaf name {
      type string;
      description "An arbitrary name for the RESTCONF listen endpoint.";
    }
    choice transport {
      mandatory true;
      description
"Selects between available transports. This is a 'choice' statement so as to support additional transport options to be augmented in."

case tls {
  if-feature tls-listen;
  container tls {
    description
    "TLS-specific listening configuration for inbound connections.";
    leaf address {
      type inet:ip-address;
      description
      "The IP address to listen on for incoming connections. The RESTCONF server will listen on all configured interfaces if no value is specified. INADDR_ANY (0.0.0.0) or INADDR6_ANY (0:0:0:0:0:0:0:0 a.k.a. ::) MUST be used when the server is to listen on all IPv4 or IPv6 addresses, respectively.";
    }
    leaf port {
      type inet:port-number;
      default 443;
      description
      "The local port number to listen on. If no value is specified, the IANA-assigned port value for 'https' (443) is used.";
    }
    uses ts:tls-server-grouping {
      refine "client-auth" {
        must 'pinned-ca-certs or pinned-client-certs';
        description
        "RESTCONF servers MUST be able to validate clients.";
      }
      augment "client-auth" {
        description
        "Augments in the cert-to-name structure, so the RESTCONF server can map TLS-layer client certificates to RESTCONF usernames.";
        container cert-maps {
          uses x509c2n:cert-to-name;
          description
          "The cert-maps container is used by a TLS-based RESTCONF server to map the RESTCONF client’s presented X.509 certificate to a RESTCONF username. If no matching and valid cert-to-name list entry can be found,
then the RESTCONF server MUST close the connection, and MUST NOT accept RESTCONF messages over it.

reference
"RFC 7407: A YANG Data Model for SNMP Configuration.";

} // end container
} // end case
} // end transport
} // end endpoint
} // end listen

container call-home {
if-feature call-home;
presence "Enables server to initiate TCP connections";
description "Configures call-home behavior";
list restconf-client {
key name;
min-elements 1;
description
"List of RESTCONF clients the RESTCONF server is to initiate call-home connections to in parallel.";
leaf name {
type string;
description
"An arbitrary name for the remote RESTCONF client.";
}
} // end container

container endpoints {
description
"Container for the list of endpoints.";
list endpoint {
key name;
min-elements 1;
ordered-by user;
description
"User-ordered list of endpoints for this RESTCONF client. Defining more than one enables high-availability.";
leaf name {
type string;
description
"An arbitrary name for this endpoint.";
}
choice transport {
mandatory true;
"Selects between available transports. This is a ‘choice’ statement so as to support additional transport options to be augmented in."

```yml
case tls {
  if-feature tls-call-home;
  container tls {
    description
    "Specifies TLS-specific call-home transport configuration.";
    leaf address {
      type inet:host;
      mandatory true;
      description
      "The IP address or hostname of the endpoint. If a domain name is configured, then the DNS resolution should happen on each usage attempt. If the DNS resolution results in multiple IP addresses, the IP addresses will be tried according to local preference order until a connection has been established or until all IP addresses have failed.";
    }
    leaf port {
      type inet:port-number;
      default 4336;
      description
      "The IP port for this endpoint. The RESTCONF server will use the IANA-assigned well-known port for 'restconf-ch-tls' (4336) if no value is specified.";
    }

    uses ts:tls-server-grouping {
      refine "client-auth" {
        must 'pinned-ca-certs or pinned-client-certs';
        description
        "RESTCONF servers MUST be able to validate clients.";
      }

      augment "client-auth" {
        description
        "Augments in the cert-to-name structure, so the RESTCONF server can map TLS-layer client certificates to RESTCONF usernames.";
        container cert-maps {
          uses x509c2n:cert-to-name;
          description
          "The cert-maps container is used by a"}
```
TLS-based RESTCONF server to map the
RESTCONF client’s presented X.509
certificate to a RESTCONF username. If
no matching and valid cert-to-name list
entry can be found, then the RESTCONF
server MUST close the connection, and
MUST NOT accept RESTCONF messages over
it.

reference
"RFC 7407: A YANG Data Model for SNMP
Configuration.";

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

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

container keep-alives {
  description
  "Configures the keep-alive policy, to proactively test the aliveness of the TLS client. An unresponsive TLS client will be dropped after approximately (max-attempts * max-wait) seconds.";
  reference
  "RFC 8071: NETCONF Call Home and RESTCONF Call Home, Section 3.1, item S6"
  leaf max-wait {
    type uint16 {
      range "1..max";
    }
    units seconds;
    default 30;
    description
    "Sets the amount of time in seconds after which if no data has been received from the TLS client, a TLS-level message will be sent to test the aliveness of the TLS client.";
  }
  leaf max-attempts {
    type uint8;
    default 3;
    description
    "Sets the maximum number of sequential keep-alive messages that can fail to obtain a response from the TLS client before assuming the TLS client is no longer alive.";
  }
}

}

case periodic-connection {
  container periodic {
    presence
    "Indicates that a periodic connection is to be maintained.";
  }
}
Permanently connect to the RESTCONF client, so that the RESTCONF client may send requests pending for the RESTCONF server. Once the connection has been closed, for whatever reason, the server will restart its timer until the next connection.

```xml
<leaf idle-timeout {
  type uint16;
  units "seconds";
  default 300; // five minutes
  description
  "Specifies the maximum number of seconds that the underlying TLS session may remain idle. A TLS session will be dropped if it is idle for an interval longer than this number of seconds. If set to zero, then the server will never drop a session because it is idle. Sessions that have a notification subscription active are never dropped.";
}

leaf reconnect-timeout {
  type uint16 {
    range "1..max";
  }
  units minutes;
  default 60;
  description
  "The maximum amount of unconnected time the RESTCONF server will wait before re-establishing a connection to the RESTCONF client. The RESTCONF server may initiate a connection to the RESTCONF client before this time if desired (e.g., to deliver a notification).";
}

container reconnect-strategy {
  description
  "The reconnection strategy directs how a RESTCONF server reconnects to a RESTCONF client after discovering its connection to the client has dropped, even if due to a reboot. The RESTCONF server starts with the specified endpoint and tries to connect to it max-attempts times before trying the next endpoint in the list (round robin).";
}
```
leaf start-with {
  type enumeration {
    enum first-listed {
      description
      "Indicates that reconnections should start with
      the first endpoint listed.";
    }
    enum last-connected {
      description
      "Indicates that reconnections should start with
      the endpoint last connected to. If no previous
      connection has ever been established, then the
      first endpoint configured is used. RESTCONF
      servers SHOULD be able to remember the last
      endpoint connected to across reboots.";
    }
  }
  default first-listed;
  description
  "Specifies which of the RESTCONF client’s endpoints
  the RESTCONF server should start with when trying
  to connect to the RESTCONF client.";
}
leaf max-attempts {
  type uint8 {
    range "1..max";
  }
  default 3;
  description
  "Specifies the number times the RESTCONF server tries
  to connect to a specific endpoint before moving on to
  the next endpoint in the list (round robin).";
}

4. Security Considerations

The YANG module defined in this document uses a grouping defined in
[I-D.ietf-netconf-tls-client-server]. Please see the Security
Considerations section in that document for concerns related that
grouping.
The YANG module defined in this document is designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

```
/: The entire data trees defined by the modules defined in this draft are sensitive to write operations. For instance, the addition or removal of references to keys, certificates, trusted anchors, etc., can dramatically alter the implemented security policy. However, no NACM annotations are applied as the data SHOULD be editable by users other than a designated ‘recovery session’.
```

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

```
NONE
```

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

```
NONE
```

5. IANA Considerations

5.1. The IETF XML Registry

This document registers two URIs in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:
5.2. The YANG Module Names Registry

This document registers two YANG modules in the YANG Module Names registry [RFC7950]. Following the format in [RFC7950], the following registrations are requested:

   name:         ietf-restconf-client
   prefix:       ncc
   reference:    RFC XXXX

   name:         ietf-restconf-server
   prefix:       ncs
   reference:    RFC XXXX

6. References

6.1. Normative References


Watsen & Wu           Expires December 6, 2018       [Page 34]
6.2. Informative References


Appendix A. Change Log

A.1. 00 to 01
- Renamed "keychain" to "keystore".

A.2. 01 to 02
- Filled in previously missing 'ietf-restconf-client' module.
- Updated the ietf-restconf-server module to accomodate new grouping 'ietf-tls-server-grouping'.

A.3. 02 to 03
- Refined use of tls-client-grouping to add a must statement indicating that the TLS client must specify a client-certificate.
- Changed restconf-client??? to be a grouping (not a container).

A.4. 03 to 04
- Added RFC 8174 to Requirements Language Section.
- Replaced refine statement in ietf-restconf-client to add a mandatory true.
- Added refine statement in ietf-restconf-server to add a must statement.
- Now there are containers and groupings, for both the client and server models.
- Now tree diagrams reference ietf-netmod-yang-tree-diagrams
- Updated examples to inline key and certificates (no longer a leafref to keystore)

A.5. 04 to 05
- Now tree diagrams reference ietf-netmod-yang-tree-diagrams
- Updated examples to inline key and certificates (no longer a leafref to keystore)
A.6.  05 to 06

- Fixed change log missing section issue.
- Updated examples to match latest updates to the crypto-types, trust-anchors, and keystore drafts.
- Reduced line length of the YANG modules to fit within 69 columns.

Acknowledgements

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

Authors’ Addresses

Kent Watsen  
Juniper Networks  
EMail: kwatsen@juniper.net

Gary Wu  
Cisco Networks  
EMail: garywu@cisco.com
Abstract

This document defines RESTCONF, HTTP2, and HTTP1.1 bindings for the transport of subscription requests and corresponding push updates. Being subscribed may be either publisher defined event streams or nodes/subtrees of YANG Datastores.

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 December 20, 2018.

Copyright Notice

Copyright (c) 2018 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 .............................................. 3
2. Terminology ............................................. 3
3. Dynamic Subscription ..................................... 3
   3.1. Transport Connectivity .............................. 4
   3.2. Discovery ........................................... 4
   3.3. RESTCONF RPCs and HTTP Status Codes ............... 4
   3.4. Call Flow for HTTP2 ................................ 6
   3.5. Call flow for HTTP1.1 .............................. 8
4. Configured Subscription ................................. 10
   4.1. Transport Connectivity .............................. 10
   4.2. Call Flow ......................................... 11
5. QoS Treatment ............................................ 12
6. Mandatory JSON and datastore support .................... 12
7. Notification Messages .................................... 12
8. YANG Tree ............................................... 12
9. YANG module ............................................ 13
10. IANA Considerations ..................................... 15
11. Security Considerations ................................. 15
12. Acknowledgments ........................................ 16
13. References .............................................. 16
   13.1. Normative References .............................. 17
   13.2. Informative References ............................ 18
Appendix A. RESTCONF over GRPC .............................. 19
Appendix B. Examples ...................................... 19
   B.1. Dynamic Subscriptions .............................. 19
      B.1.1. Establishing Dynamic Subscriptions .......... 19
      B.1.2. Modifying Dynamic Subscriptions ............. 22
      B.1.3. Deleting Dynamic Subscriptions .............. 23
   B.2. Configured Subscriptions .......................... 24
      B.2.1. Creating Configured Subscriptions .......... 25
      B.2.2. Modifying Configured Subscriptions .......... 27
      B.2.3. Deleting Configured Subscriptions .......... 29
   B.3. Subscription State Notifications .................. 30
      B.3.1. subscription-started and subscription-modified 30
      B.3.2. subscription-completed, subscription-resumed, and replay-complete 31
      B.3.3. subscription-terminated and subscription-suspended 31
Appendix C. Changes between revisions ........................ 32
Authors’ Addresses .......................................... 33
1. Introduction

Mechanisms to support event subscription and push are defined in [I-D.draft-ietf-netconf-subscribed-notifications]. Enhancements to [I-D.draft-ietf-netconf-subscribed-notifications] which enable YANG datastore subscription and push are defined in [I-D.ietf-netconf-yang-push]. This document provides a transport specification for these protocols over RESTCONF [RFC8040] and HTTP. Driving these requirements is [RFC7923].

The streaming of notifications encapsulating the resulting information push can be done with either HTTP1.1 [RFC7231] or HTTP2 [RFC7540].

2. Terminology

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

The following terms use the definitions from [I-D.draft-ietf-netconf-subscribed-notifications]: configured subscription, dynamic subscription, event stream, notification message, publisher, receiver, subscriber, and subscription.

Other terms reused include datastore, which is defined in [RFC8342], and HTTP2 stream which maps to the definition of "stream" within [RFC7540], Section 2.

[ note to the RFC Editor - please replace XXXX within this document with the number of this document ]

3. Dynamic Subscription

This section provides specifics on how to establish and maintain dynamic subscriptions over HTTP 1.1 and HTTP2 via signaling messages transported over RESTCONF [RFC8040]. Subscribing to event streams is accomplished in this way via a RESTCONF POST into RPCs defined within [I-D.draft-ietf-netconf-subscribed-notifications] Section 2.4. YANG datastore subscription is accomplished via augmentations to [I-D.draft-ietf-netconf-subscribed-notifications] as described within [I-D.ietf-netconf-yang-push] Section 4.4.

Common across all HTTP based dynamic subscriptions is that a POST needs to be made against a specific URI on the Publisher. Subscribers cannot pre-determine the URI against which a subscription might exist on a publisher, as the URI will only exist after the "establish-subscription" has been accepted. The subscription URI
will be determined and sent as part of the response to the "establish-subscription", and a subsequent POST to this URI will be done in order to start the flow of notification messages back to the subscriber. A subscription does not move to the active state as per Section 2.4.1. of [I-D.draft-ietf-netconf-subscribed-notifications] until the POST is received.

3.1. Transport Connectivity

For a dynamic subscription, where an HTTP client session doesn’t already exist, a new client session is initiated from the subscriber. If the subscriber is unsure if HTTP2 is supported by the publisher, HTTP1.1 will be used for initial messages, and these messages will include an HTTP version upgrade request as per [RFC7230], Section 6.7. If a publisher response indicates that HTTP2 is supported, HTTP2 will be used between subscriber and publisher for future HTTP interactions as per [RFC7540].

A subscriber SHOULD establish the HTTP session over TLS [RFC5246] in order to secure the content in transit.

Without the involvement of additional protocols, neither HTTP1.1 nor HTTP2 sessions by themselves allow for a quick recognition of when the communication path has been lost with the publisher. Where quick recognition of the loss of a publisher is required, a subscriber SHOULD connect over TLS [RFC5246], and use a TLS heartbeat [RFC6520] to track HTTP session continuity. In the case where a TLS heartbeat is included, it should be sent just from receiver to publisher. Loss of the heartbeat MUST result in any subscription related TCP sessions between those endpoints being torn down. A subscriber can then attempt to re-establish.

3.2. Discovery

Subscribers can learn what event streams a RESTCONF server supports by querying the "streams" container of ietf-subscribed-notification.yang. Subscribers can learn what datastores a RESTCONF server supports by following [I-D.draft-ietf-netconf-nmda-restconf].

3.3. RESTCONF RPCs and HTTP Status Codes

Specific HTTP responses codes as defined in [RFC7231] section 6 will indicate the result of RESTCONF RPC requests with publisher. An HTTP status code of 200 is the proper response to any successful RPC defined within [I-D.draft-ietf-netconf-subscribed-notifications] or [I-D.ietf-netconf-yang-push].
If a publisher fails to serve the RPC request for one of the reasons indicated in [I-D.draft-ietf-netconf-subscribed-notifications] Section 2.4.6 or [I-D.ietf-netconf-yang-push] Appendix A, this will be indicated by "406" status code transported in the HTTP response.

When a "406" status code is returned, the RPC reply MUST include an "rpc-error" element per [RFC8040] Section 7.1 with the following parameter values:

- an "error-type" node of "application".
- an "error-tag" node of "operation-failed".
- an "error-app-tag" node with the value being a string that corresponds to an identity associated with the error, as defined in [I-D.draft-ietf-netconf-subscribed-notifications] section 2.4.6 for general subscriptions, and [I-D.ietf-netconf-yang-push] Appendix A.1, for datastore subscriptions. The tag to use depends on the RPC for which the error occurred. Viable errors for different RPCs are as follows:

<table>
<thead>
<tr>
<th>RPC</th>
<th>select an identity with a base</th>
</tr>
</thead>
<tbody>
<tr>
<td>establish-subscription</td>
<td>establish-subscription-error</td>
</tr>
<tr>
<td>modify-subscription</td>
<td>modify-subscription-error</td>
</tr>
<tr>
<td>delete-subscription</td>
<td>delete-subscription-error</td>
</tr>
<tr>
<td>kill-subscription</td>
<td>kill-subscription-error</td>
</tr>
<tr>
<td>resynch-subscription</td>
<td>resynch-subscription-error</td>
</tr>
</tbody>
</table>

Each error identity will be inserted as the "error-app-tag" using JSON encoding following the form <modulename>:<identityname>. An example of such as valid encoding would be "ietf-subscribed-notifications:no-such-subscription".

- In case of error responses to an "establish-subscription" or "modify-subscription" request there is the option of including an "error-info" node. This node may contain hints for parameter settings that might lead to successful RPC requests in the future. Following are the yang-data structures which may be returned:
establish-subscription returns hints in yang-data structure
---------------------- ------------------------------------
target: event stream establish-subscription-stream-error-info
target: datastore establish-subscription-datastore-error-info

modify-subscription returns hints in yang-data structure
---------------------- ------------------------------------
target: event stream modify-subscription-stream-error-info
target: datastore modify-subscription-datastore-error-info

The yang-data included within "error-info" SHOULD NOT include the optional leaf "error-reason", as such a leaf would be redundant with information that is already placed within the "error-app-tag".

In case of an rpc error as a result of a "delete-subscription", a "kill-subscription", or a "resynch-subscription" request, no "error-info" needs to be included, as the "subscription-id" is the only RPC input parameter and no hints regarding this RPC input parameters need to be provided.

Note that "error-path" does not need to be included with the "rpc-error" element, as subscription errors are generally not associated with nodes in the datastore but with the choice of RPC input parameters.

3.4. Call Flow for HTTP2

Requests to [I-D.draft-ietf-netconf-subscribed-notifications] or [I-D.ietf-netconf-yang-push] augmented RFCs are sent on one or more HTTP2 streams indicated by (a) in Figure 1. A successful "establish-subscription" will result in an RPC response returned with both a subscription identifier which uniquely identifies a subscription, as well as a URI which uniquely identifies the location of subscription on the publisher. This URI is defined via the "uri" leaf the Data Model in Section 9.

An HTTP POST is then sent on a logically separate HTTP2 stream (b) to the URI on the publisher. This initiates to initiate the flow of notification messages which are sent in HTTP Data frames as a response to the POST. In the case below, a newly established subscription has its associated notification messages pushed over HTTP2 stream (7). These notification messages are placed into a HTTP2 Data frame (see [RFC7540] Section 6.1).
Additional requirements for dynamic subscriptions over HTTP2 include:

- A unique HTTP2 stream MAY be used for each subscription.
- A single HTTP2 stream MUST NOT be used for subscriptions with different DSCP values.
- All subscription state notifications from a publisher MUST be returned in a separate HTTP Data frame within the HTTP2 stream used by the subscription to which the state change refers.
- In addition to an RPC response for a "modify-subscription" RPC traveling over (a), a "subscription-modified" state change notification must be sent within HTTP2 stream (b). This allows the receiver to know exactly when the new terms of the
subscription have been applied to the notification messages. See
arrow (c).

- Additional RPCs for a particular subscription MUST NOT use the
  HTTP2 stream currently providing notification messages
  subscriptions.

- An HTTP end of stream message MUST not be sent until all
  subscriptions using that HTTP2 stream have completed.

3.5. Call flow for HTTP1.1

The call flow is defined in Figure 2. Requests to
[I-D.draft-ietf-netconf-subscribed-notifications] or
[I-D.ietf-netconf-yang-push] augmented RPCs are sent on a TCP
connection indicated by (a). A successful "establish-subscription"
will result in an RPC response returned with both a subscription
identifier which uniquely identifies a subscription, as well as a URI
which uniquely identifies the location of subscription on the
publisher (b). This URI is defined via the "uri" leaf the Data Model
in Section 9.

An HTTP POST is then sent on a logically separate TCP connection (b)
to the URI on the publisher. This initiates to initiate the flow of
notification messages which are sent in SSE [W3C-20150203] as a
response to the POST.
Additional requirements for dynamic subscriptions over HTTP1.1 include:

- All subscription state notifications from a publisher MUST be returned in a separate SSE message used by the subscription to which the state change refers.

- Subscription RPCs MUST NOT use the TCP connection currently providing notification messages for that subscription.

- In addition to an RPC response for a "modify-subscription" RPC traveling over (a), a "subscription-modified" state change notification must be sent within stream (b). This allows the receiver to know exactly when the new terms of the subscription have been applied to the notification messages. See arrow (c).
Open question, should we just eliminate this possibility of HTTP1.1 for subscriptions? It would make the design simpler.

4. Configured Subscription

With a configured subscription, all information needed to establish a secure relationship with that receiver is available on the publisher. With this information, the publisher will establish a secure transport connection with the receiver and then begin pushing notification messages to the receiver. Since RESTCONF might not exist on the receiver, it is not desirable to require that subscribed content be pushed with any dependency on RESTCONF. Therefore in place of RESTCONF, an HTTP2 Client connection must be established with an HTTP2 Server located on the receiver. Notification messages will then be sent as part of an extended HTTP POST to the receiver.

4.1. Transport Connectivity

Configured subscriptions MUST only be connected over HTTP2 via a client session initiated from the publisher. Following are the conditions which MUST be met before establishing a new HTTP2 connection with a receiver:

- a configured subscription has a receiver in the connecting state as described in [I-D.draft-ietf-netconf-subscribed-notifications], section 2.5.1.,
- the transport configured for that subscription is HTTP2,
- there are state change notifications or notification messages pending for that receiver, and
- no HTTP2 transport session exists to that receiver,

If the above conditions are met, then the publisher MUST initiate a transport session via RESTCONF call home [RFC8071], section 4.1 to that receiver. HTTP2 only communications must be used as per [RFC7540], Section 3.3 when the HTTP session over TLS [RFC5246]. and [RFC7540], Section 3.4 when transporting cleartext over TCP. Note that a subscriber SHOULD establish over TLS in order to secure the content in transit.

If the RESTCONF call home fails because the publisher receives receiver credentials which are subsequently declined per [RFC8071], Section 4.1, step S5 authentication, then that receiver MUST be placed into the timeout state.
If the call home fails to establish for any other reason, the publisher MUST NOT progress the receiver to the active state. Additionally, the publisher SHOULD place the receiver into the timeout state after a predetermined number of either failed call home attempts or remote transport session termination by the receiver.

4.2. Call Flow

With HTTP2 connectivity established, a POST of each new "subscription-started" state change notification messages will be addressed to HTTP augmentation code on the receiver capable of accepting and acknowledging to subscription state change notifications. Until the "HTTP 200 OK" at point (c) of Figure 3 for each the "subscription-started" state change notification, a publisher MUST NOT progress the receiver to the active state. In other words, is at point (c) which indicates that the receiver is ready for the delivery of subscribed content. At this point a notification-messages including subscribed content may be placed onto an HTTP2 stream for that subscription.

```
+------------+                                 +------------+
|  Receiver  |                                 | Publisher  |
|HTTP2 Stream|                                 |HTTP2 Stream|
|  (a)  (b)  |                                 |  (a)  (b)  |
+------------+                                 +------------+
 |HTTP Post Headers, Data (subscription-started) |
<---------------------------------------------|
 | HTTP 200 OK                                  |
|-------------------------------------------->(c)
 |HTTP Post Headers, Data (notif-message)      |
<---------------------------------------------|
 | HTTP Data (notif-message)                   |
|-------------------------------------------->(c)
 |HTTP Post Headers, Data (notif-message)      |
<---------------------------------------------|
 | HTTP Data (notif-message)                   |
<-------------------------------------------->(c)
 |HTTP Data (sub-terminated)                   |
<---------------------------------------------|
 |HTTP 200 OK                                  |
|-------------------------------------------->(c)
```

Figure 3: Configured over HTTP2

Additional requirements for configured subscriptions over HTTP2 include:

- A unique HTTP2 stream MAY be used for each subscription.
- A single HTTP2 stream MUST NOT be used for subscriptions with different DSCP values.
5. QoS Treatment

To meet subscription quality of service promises, the publisher MUST take any existing subscription "dscp" and apply it to the DSCP marking in the IP header.

In addition, where HTTP2 transport is available to a notification message queued for transport to a receiver, the publisher MUST:

- take any existing subscription "priority" and copy it into the HTTP2 stream priority,
- take any existing subscription "dependency" and map the HTTP2 stream for the parent subscription into the HTTP2 stream dependency.

6. Mandatory JSON and datastore support

A publisher supporting [I-D.ietf-netconf-yang-push] MUST support the "operational" datastore as defined by [RFC8342].

The "encode-json" feature of [I-D.draft-ietf-netconf-subscribed-notifications] is mandatory to support. This indicates that JSON is a valid encoding for RPCs, state change notifications, and subscribed content.

7. Notification Messages

Notification messages transported over HTTP will be encoded using one-way operation schema defined within [RFC5277], section 4.

8. YANG Tree

The YANG model defined in Section 9 has one leaf augmented into four places of [I-D.draft-ietf-netconf-subscribed-notifications], plus two identities. As the resulting full tree is large, it will only be inserted at later stages of this document.
9. YANG module

This module references [I-D.draft-ietf-netconf-subscribed-notifications].

<CODE BEGINS> file "ietf-http-subscribed-notifications@2018-06-11.yang"

module ietf-http-subscribed-notifications {
  yang-version 1.1;
  namespace
  prefix hsn;

  import ietf-subscribed-notifications {
    prefix sn;
  }
  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>
    Editor:   Eric Voit
              <mailto:evoit@cisco.com>
    Editor:   Alexander Clemm
              <mailto:ludwig@clemm.org>
    Editor:   Reshad Rahman
              <mailto:rrahman@cisco.com>";

  description
    "Defines HTTP variants as a supported transports for subscribed
     event notifications."

  Copyright (c) 2018 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

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

revision 2018-06-11 {
    description
    "Initial version";
    reference
    "RFC XXXX: RESTCONF and HTTP Transport for Event Notifications";
}

identity http2 {
    base sn:transport;
    base sn:inline-address;
    base sn:configurable-encoding;
    description
    "HTTP2 is used a transport for notification messages and state change notifications."
}

identity http1.1 {
    base sn:transport;
    base sn:inline-address;
    base sn:configurable-encoding;
    description
    "HTTP1.1 is used a transport for notification messages and state change notifications."
}

grouping uri {
    description
    "Provides a reusable description of a URI.";
    leaf uri {
        type inet:uri;
        config false;
        description
        "Location of a subscription specific URI on the publisher."
    }
}

augment "/sn:establish-subscription/sn:output" {
    description
    "This augmentation allows HTTP specific parameters for a response to a publisher's subscription request.";
    uses uri;
}

augment "/sn:subscriptions/sn:subscription" {
    description
    "This augmentation allows HTTP specific parameters to be
exposed for a subscription.
}

uses uri;
}

augment "/sn:subscription-started" {
  description
    "This augmentation allows HTTP specific parameters to be included part of the notification that a subscription has started.";
  uses uri;
}

augment "/sn:subscription-modified" {
  description
    "This augmentation allows HTTP specific parameters to be included part of the notification that a subscription has been modified.";
  uses uri;
}

<CODE ENDS>

10.  IANA Considerations

This document registers the following namespace URI in the "IETF XML Registry" [RFC3688]:

Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

This document registers the following YANG module in the "YANG Module Names" registry [RFC6020]:

Name: ietf-http-subscribed-notifications
Prefix: hsn
Reference: RFC XXXX: RESTCONF and HTTP Transport for Event Notifications

11.  Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management transports such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest
RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The one new data node introduced in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to this data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

Container: "/subscriptions"

- **"uri":** leaf will show where subscribed resources might be located on a publisher. Access control must be set so that only someone with proper access permissions, and perhaps even HTTP session has the ability to access this resource.

One or more publishers of configured subscriptions could be used to overwhelm a receiver which doesn’t even support subscriptions. There are two protections needing support on a publisher. First, notification messages for configured subscriptions MUST only be transmittable over encrypted transports. Clients which do not want pushed content need only terminate or refuse any transport sessions from the publisher. Second, the HTTP transport augmentation on the receiver must send an HTTP 200 OK to a subscription started notification before the publisher starts streaming any subscribed content.

One or more publishers could overwhelm a receiver which is unable to control or handle the volume of Event Notifications received. In deployments where this might be a concern, HTTP2 transport such as HTTP2) should be selected.

The NETCONF Authorization Control Model [RFC6536] SHOULD be used to control and restrict authorization of subscription configuration.

12. Acknowledgments

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from: Ambika Prasad Tripathy, Alberto Gonzalez Prieto, Susan Hares, Tim Jenkins, Balazs Lengyel, Kent Watsen, Michael Scharf, and Guangying Zheng.

13. References
13.1. Normative References


[I-D.draft-ietf-netconf-subscribed-notifications]

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


13.2. Informative References

[W3C-20150203]
"Server-Sent Events, World Wide Web Consortium CR CR-
eventsource-2012111", February 2015,
<https://www.w3.org/TR/2015/REC-eventsource-20150203/>.

[I-D.draft-ietf-netconf-netconf-event-notifications]
Clemm, Alexander., Voit, Eric., Gonzalez Prieto, Alberto.,
Nilsen-Nygaard, E., and A. Tripathy, "NETCONF support for
event notifications", May 2018,
<https://datatracker.ietf.org/doc/
draft-ietf-netconf-netconf-event-notifications/>.

Voit, et al.            Expires December 20, 2018
[Page 18]
Appendix A. RESTCONF over GRPC

An initial goal for this document was to support [GRPC] transport seamlessly without any mapping or extra layering. However there is an incompatibility of RESTCONF and GRPC. RESTCONF uses HTTP GET, and GRPC uses HTTP2’s POST rather than GET. As GET is used across RESTCONF for things like capabilities exchange, a seamless mapping depends on specification changes outside the scope of this document. If/when GRPC supports GET, or RESTCONF is updated to support POST, this should be revisited. It is hoped that the resulting fix will be transparent to this document.

Appendix B. Examples

This section is non-normative. To allow easy comparison, this section mirrors the functional examples shown with NETCONF over XML within [I-D.draft-ietf-netconf-netconf-event-notifications]. In addition, HTTP2 vs HTTP1.1 headers are not shown as the contents of the JSON encoded objects are identical within.

B.1. Dynamic Subscriptions

B.1.1. Establishing Dynamic Subscriptions

The following figure shows two successful "establish-subscription" RPC requests as per [I-D.draft-ietf-netconf-subscribed-notifications]. The first request...
is given a subscription identifier of 22, the second, an identifier of 23.

```
+-----------+                  +-----------+
| Subscriber |                  | Publisher |
+-----------+                  +-----------+

establish-subscription
------------------------------>  (a)
HTTP 200 OK, id#22, URI#1
<-------------------------------
POST (URI#1)

establish-subscription
------------------------------>  (b)
HTTP 200 OK, notif-mesg (id#22)
<-------------------------------
POST (URI#1)

establish-subscription
------------------------------>  (c)
HTTP 200 OK, id#23, URI#2
<-------------------------------
POST (URI#2)

notif-mesg (id#22)
<-------------------------------
HTTP 200 OK, notif-mesg (id#23)
<-------------------------------
```

Figure 4: Multiple subscriptions over RESTCONF/HTTP

To provide examples of the information being transported, example messages for interactions in Figure 4 are detailed below:

```
POST /restconf/operations/subscriptions:establish-subscription
{
  "ietf-subscribed-notifications:input": {
    "stream": "NETCONF",
    "stream-xpath-filter": "/ex:foo/",
    "dscp": "10"
  }
}
```

Figure 5: establish-subscription request (a)
As publisher was able to fully satisfy the request, the publisher sends the subscription identifier of the accepted subscription, and the URI:

HTTP status code - 200

```
{  
  "identifier": "22",
  "uri": "/subscriptions/22"
}
```

Figure 6: establish-subscription success (b)

Upon receipt of the successful response, the subscriber POSTs to the provided URI to start the flow of notification messages. When the publisher receives this, the subscription is moved to the active state (c).

POST /restconf/operations/subscriptions/22

Figure 7: establish-subscription subsequent POST

While not shown in Figure 4, if the publisher had not been able to fully satisfy the request, or subscriber has no authorization to establish the subscription, the publisher would have sent an RPC error response. For instance, if the "dscp" value of 10 asserted by the subscriber in Figure 5 proved unacceptable, the publisher may have returned:

HTTP status code - 406

```
{  "ietf-restconf:errors" : {  
  "error" : [  
  {  
  "error-type": "application",
  "error-tag": "operation-failed",
  "error-severity": "error",
  "error-app-tag": "ietf-subscribed-notifications:dscp-unavailable"
  ]  
  ]  
}
```

Figure 8: an unsuccessful establish subscription
The subscriber can use this information in future attempts to establish a subscription.

B.1.2. Modifying Dynamic Subscriptions

An existing subscription may be modified. The following exchange shows a negotiation of such a modification via several exchanges between a subscriber and a publisher. This negotiation consists of a failed RPC modification request/response, followed by a successful one.

```
+------------+                 +-----------+
| Subscriber |                 | Publisher |
+------------+                 +-----------+
|                              |           |
| notification message (id#23) |           |
|-------------------------------|           |
| modify-subscription (id#23)   | (d)       |
|-------------------------------| (e)       |
| HTTP 406 error (with hint)    |           |
|-------------------------------|           |
| modify-subscription (id#23)   |           |
|-------------------------------|           |
| HTTP 200 OK                   |           |
|-------------------------------|           |
| notif-mesg (id#23)            |           |
```

Figure 9: Interaction model for successful subscription modification

If the subscription being modified in Figure 9 is a datastore subscription as per [I-D.ietf-netconf-yang-push], the modification request made in (d) may look like that shown in Figure 10. As can be seen, the modifications being attempted are the application of a new xpath filter as well as the setting of a new periodic time interval.
POST /restconf/operations/subscriptions:modify-subscription
{
  "ietf-subscribed-notifications:input": {
    "identifier": "23",
    "ietf-yang-push:datastore-xpath-filter": "/interfaces-state/interface/oper-status"
    "ietf-yang-push:periodic": {
      "ietf-yang-push:period": "500"
    }
  }
}

Figure 10: Subscription modification request (c)

If the publisher can satisfy both changes, the publisher sends a positive result for the RPC. If the publisher cannot satisfy either of the proposed changes, the publisher sends an RPC error response (e). The following is an example RPC error response for (e) which includes a hint. This hint is an alternative time period value which might have resulted in a successful modification:

HTTP status code - 406
{
  "ietf-restconf:errors" : {
    "error" : [
      "error-type": "application",
      "error-tag": "operation-failed",
      "error-severity": "error",
      "error-app-tag": "ietf-yang-push:period-unsupported",
      "error-info": {
        "ietf-yang-push":
        "modify-subscription-datastore-error-info": {
          "period-hint": "3000"
        }
      }
    ]
  }
}

Figure 11: Modify subscription failure with Hint (e)

B.1.3. Deleting Dynamic Subscriptions

The following demonstrates deleting a subscription. This subscription may have been to either a stream or a datastore.
POST /restconf/operations/subscriptions:delete-subscription
{
   "delete-subscription": {
      "identifier": "22"
   }
}

Figure 12: Delete subscription

If the publisher can satisfy the request, the publisher replies with success to the RPC request.

If the publisher cannot satisfy the request, the publisher sends an error-rpc element indicating the modification didn’t work. Figure 13 shows a valid response for existing valid subscription identifier, but that subscription identifier was created on a different transport session:

   HTTP status code - 406
   
   {  
   "ietf-restconf:errors" : {  
   "error" : [  
   "error-type": "application",  
   "error-tag": "operation-failed",  
   "error-severity": "error",  
   "error-app-tag":  
   "ietf-subscribed-notifications:no-such-subscription"
   ]
   }
   }

Figure 13: Unsuccessful delete subscription

B.2. Configured Subscriptions

Configured subscriptions may be established, modified, and deleted using configuration operations against the top-level subtree of [I-D.draft-ietf-netconf-subscribed-notifications] or [I-D.ietf-netconf-yang-push].

In this section, we present examples of how to manage the configuration subscriptions using a HTTP2 client.
B.2.1. Creating Configured Subscriptions

For subscription creation via configuration operations, a RESTCONF client may send:

```
POST /restconf/operations/subscriptions/
{
   "edit-config": {
      "target": {
         "running": null
      },
      "default-operation": "none",
      "config": {
         "subscriptions": {
            "subscription": {
               "identifier": "22",
               "transport": "HTTP2",
               "stream": "NETCONF",
               "receivers": {
                  "receiver": {
                     "name": "receiver1",
                     "address": "1.2.3.4"
                  }
               }
            }
         }
      }
   }
}
```

Figure 14: Create a configured subscription

If the request is accepted, the publisher will indicate this. If the request is not accepted because the publisher cannot serve it, no configuration is changed. In this case the publisher may reply:
HTTP status code - 406

```
{
    "ietf-restconf:errors": {
        "error": [
            {
                "error-type": "application",
                "error-tag": "resource-denied",
                "error-severity": "error",
                "error-message": {
                    "@lang": "en",
                    "#text": "Temporarily the publisher cannot serve this subscription due to the current workload."
                }
            }
        ]
    }
}
```

Figure 15: Response to a failed configured subscription establishment

After a subscription has been created and been verified as VALID, HTTP2 connectivity to each receiver will be established if that connectivity does not already exist.

The following figure shows the interaction model for the successful creation of a configured subscription.
B.2.2. Modifying Configured Subscriptions

Configured subscriptions can be modified using configuration operations against the top-level container "/subscriptions".

For example, the subscription established in the previous section could be modified as follows, here adding a second receiver:
POST /restconf/operations/subscriptions

{
  "edit-config": {
    "target": {
      "running": null
    },
    "config": {
      "subscriptions": {
        "subscription": {
          "identifier": "1922",
          "receivers": {
            "receiver": {
              "name": "receiver2",
              "address": "1.2.3.5"
            }
          }
        }
      }
    }
  }
}

Figure 17: Modify configured subscription

If the request is accepted, the publisher will indicate success. The result is that the interaction model described in Figure 16 may be extended as follows.
B.2.3. Deleting Configured Subscriptions

Configured subscriptions can be deleted using configuration operations against the top-level container "/subscriptions". Deleting the subscription above would result in the following flow impacting all active receivers.
B.3. Subscription State Notifications

A publisher will send subscription state notifications according to the definitions within [I-D.draft-ietf-netconf-subscribed-notifications]).

B.3.1. subscription-started and subscription-modified

A "subscription-started" encoded in JSON would look like:

```
{
    "ietf-restconf:notification" : {
        "eventType": "2007-09-01T10:00:00Z",
        "ietf-subscribed-notifications:subscription-started": {
            "identifier": "39",
            "transport": "HTTP2",
            "stream-xpath-filter": "//ex:foo",
            "stream": {
                "ietf-netconf-subscribed-notifications" : "NETCONF"
            }
        }
    }
}
```

Figure 20: subscription-started subscription state notification
The "subscription-modified" is identical to Figure 20, with just the word "started" being replaced by "modified".

B.3.2. subscription-completed, subscription-resumed, and replay-complete

A "subscription-completed" would look like:

```
{
  "ietf-restconf:notification": {
    "eventTime": "2007-09-01T10:00:00Z",
    "ietf-subscribed-notifications:subscription-completed": {
      "identifier": "39",
    }
  }
}
```

Figure 21: subscription-completed notification in JSON

The "subscription-resumed" and "replay-complete" are virtually identical, with "subscription-completed" simply being replaced by "subscription-resumed" and "replay-complete".

B.3.3. subscription-terminated and subscription-suspended

A "subscription-terminated" would look like:

```
{
  "ietf-restconf:notification": {
    "eventTime": "2007-09-01T10:00:00Z",
    "ietf-subscribed-notifications:subscription-terminated": {
      "identifier": "39",
      "error-id": "suspension-timeout"
    }
  }
}
```

Figure 22: subscription-terminated subscription state notification

The "subscription-suspended" is virtually identical, with "subscription-terminated" simply being replaced by "subscription-suspended".

Appendix C. Changes between revisions

(To be removed by RFC editor prior to publication)

v05 - v06
o JSON examples updated by Reshad.

v04 - v05
o Error mechanisms updated to match embedded RESTCONF mechanisms
o Restructured format and sections of document.
o Added a YANG data model for HTTP specific parameters.
o Mirrored the examples from the NETCONF transport draft to allow easy comparison.

v03 - v04
o Draft not fully synched to new version of subscribed-notifications yet.
o References updated

v02 - v03
o Event notification reframed to notification message.

v01 - v02
o Removed sections now redundant with [I-D.draft-ietf-netconf-subscribed-notifications] and [I-D.ietf-netconf-yang-push] such as: mechanisms for subscription maintenance, terminology definitions, stream discovery.

o 3rd party subscriptions are out-of-scope.

o SSE only used with RESTCONF and HTTP1.1 dynamic subscriptions

o Timeframes for event tagging are self-defined.

o Clean-up of wording, references to terminology, section numbers.
- Removed the ability for more than one subscription to go to a single HTTP2 stream.
- Updated call flows. Extensively.
- SSE only used with RESTCONF and HTTP1.1 dynamic subscriptions
- HTTP is not used to determine that a receiver has gone silent and is not Receiving Event Notifications
- Many clean-ups of wording and terminology

Authors’ Addresses

Eric Voit
Cisco Systems
Email: evoit@cisco.com

Reshad Rahman
Cisco Systems
Email: rrahman@cisco.com

Einar Nilsen-Nygaard
Cisco Systems
Email: einarnn@cisco.com

Alexander Clemm
Huawei
Email: ludwig@clemm.org

Andy Bierman
YumaWorks
Email: andy@yumaworks.com
Abstract

This document describes a YANG library that provides information about the YANG modules, datastores, and datastore schemas used by a network management server. Simple caching mechanisms are provided to allow clients to minimize retrieval of this information. This version of the YANG library supports the Network Management Datastore Architecture by listing all datastores supported by a network management server and the schema that is used by each of these datastores.

This document obsoletes RFC 7895.

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 11, 2018.
1. Introduction

There is a need for a standard mechanism to expose which YANG modules [RFC7950], datastores and datastore schemas [RFC8342] are in use by a network management server.

This document defines the YANG module "ietf-yang-library" that provides this information. This version of the YANG library is compatible with the Network Management Datastore Architecture (NMDA) [RFC8342]. The previous version of the YANG library, defined in [RFC7895], is not compatible with the NMDA since it assumes that all datastores have exactly the same schema. This is not necessarily true in the NMDA since dynamic configuration datastores may have their own datastore schema. Furthermore, the operational state
datastore may support non-configurable YANG modules in addition to the YANG modules supported by conventional configuration datastores.

The old YANG library definitions have been retained (for backwards compatibility reasons) but the definitions have been marked as deprecated. For backwards compatibility, an NMDA-supporting server SHOULD populate the deprecated "/modules-state" tree in a backwards-compatible manner. The new "/yang-library" tree would be ignored by legacy clients, while providing all the data needed for NMDA-aware clients, which would themselves ignore the "/modules-state" tree. The recommended approach to populate "/modules-state" is to report the schema for YANG modules that are configurable via conventional datastores and for which config false data nodes are returned via a NETCONF <get> operation, or equivalent.

The YANG library information can be different on every server and it can change at runtime or across a server reboot. If a server implements multiple network management protocols to access the server’s datastores, then each such protocol may have its own conceptual instantiation of the YANG library.

If a large number of YANG modules are utilized by a server, then the YANG library contents can be relatively large. Since the YANG library contents changes very infrequently, it is important that clients be able to cache the YANG library contents and easily identify whether their cache is out of date.

All NETCONF servers supporting YANG 1.1 [RFC7950] are required to support YANG Library (see Section 5.6.4 of RFC 7950). NETCONF servers implementing the NETCONF extensions to support the NMDA [I-D.ietf-netconf-nmda-netconf] MUST implement at least the version of the YANG library defined in this document. Similarly, all RESTCONF servers are required to support YANG Library (see Section 10 of RFC 8040). RESTCONF servers implementing the RESTCONF extensions to support the NMDA [I-D.ietf-netconf-nmda-restconf] MUST implement at least the version of the YANG library defined in this document.

1.1. 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are defined in [RFC7950]:

- module
This document uses the phrase "implementing a module" as defined in [RFC7950] Section 5.6.5.

The following terms are defined in [RFC8342]:

- datastore
- datastore schema
- configuration
- configuration datastore
- conventional configuration
- conventional configuration datastore
- operational state
- operational state datastore
- dynamic configuration datastore
- client and server

The following terms are used within this document:

- YANG library: A collection of YANG modules, submodules, datastores, and datastore schemas used by a server.
- YANG library checksum: A server-generated checksum of the contents of the YANG library.

Tree diagrams used in this document use the notation defined in [RFC8340].

2. Objectives

The following information is needed by a client application (for each YANG module in the library) to fully utilize the YANG data modeling language:

- name: The name of the YANG module.
o revision: Each YANG module and submodule within the library SHOULD have a revision. This is derived from the most recent revision statement within the module or submodule.

o submodule list: The name, and if defined, revision of each submodule used by the module must be identified.

o feature list: The name of each YANG feature supported by the server, in a given datastore schema, must be identified.

o deviation list: The name of each YANG module with deviation statements affecting a given YANG module, in a given datastore schema, must be identified.

In addition, the following information is needed by a client application for each datastore supported by a server:

o identity: The YANG identity for the datastore.

o schema: The schema (i.e., the set of modules) implemented by the datastore.

In order to select one out of several possible data model designs, the following criteria were used:

1. Efficient for a client to consume. Since the size of the YANG library can be quite large, it should be possible for clients to cache the YANG library information.

2. A dynamic configuration datastore must be able to implement a module or feature that is not implemented in the conventional configuration datastores.

3. It must be possible to NOT implement a module or feature in <operational>, even if it is implemented in some other datastore. This is required for transition purposes; a server that wants to implement <operational> should not have to implement all modules at once.

4. A given module can only be implemented in one revision in all datastores. If a module is implemented in more than one datastores, the same revision is implemented in all these datastores.

5. Multiple revisions can be used for import, if import-by revision is used.
6. Make it possible to use the YANG library by schema mount
   [I-D.ietf-netmod-schema-mount].

3. YANG Library Data Model

The "ietf-yang-library" YANG module provides information about the
modules, submodules, datastores, and datastore schemas supported by a
server. All data nodes in "ietf-yang-library" are "config false",
and thus only accessible in the operational state datastore.

```
+-----------+
| datastore |
+-----------+
  | has a V   |
    +-----------+
    | datastore | union of | module | consists of |
    | schema | ---------> | set | -------------> | modules + |
    +-----------+         +-------------------> | submodules |
```

Figure 1

The conceptual model of the YANG library is depicted in Figure 1.
Following the NMDA, every datastore has an associated datastore
schema. A datastore schema is a union of module sets and every
module set is a collection of modules and submodules, including the
modules and submodules used for imports. Note that multiple
datastores may refer to the same datastore schema. Furthermore, it
is possible that individual datastore schemas share module sets. A
common use case is the operational state datastore schema which is a
superset of the schema used by conventional configuration datastores.

Below is the YANG Tree Diagram for the "ietf-yang-library" module,
excluding the deprecated "modules-state" tree:
The "/yang-library" container holds the entire YANG library. The container has the following child nodes:

- The "/yang-library/module-set" contains entries representing module sets. The list "/yang-library/module-set/module" enumerates the modules that belong to the module set. A module is listed together with its submodules (if any), a set of features, and any deviation modules. The list "/yang-library/module-set/import-only-module" lists all modules (and their submodules) used only for imports.
The "/yang-library/schema" list contains an entry for each datastore schema supported by the server. All conventional configuration datastores use the same "schema" list entry. A dynamic configuration datastore may use a different datastore schema from the conventional configuration datastores, and hence may require a separate "schema" entry. A "schema" entry has a leaf-list of references to entries in the "module-set" list. The schema consists of the union of all modules in all referenced module sets.

The "/yang-library/datastore" list contains one entry for each datastore supported by the server, and it identifies the datastore schema associated with a datastore via a reference to an entry in the "schema" list. Each supported conventional configuration datastore has a separate entry, pointing to the same "schema" list element.

The "/yang-library/checksum" leaf contains the YANG library checksum, which is a unique implementation-specific identifier representing the current information in the YANG library on a specific server. The value of this leaf MUST change whenever the information in the YANG library changes. There is no requirement that the same information always results in the same "checksum" value. This leaf allows a client to fetch all schema information once, cache it, and only refetch it if the value of this leaf has been changed. If the value of this leaf changes, the server also generates a "yang-library-update" notification.

Note that for a NETCONF server implementing the NETCONF extensions to support the NMDA [I-D.ietf-netconf-nmda-netconf], a change of the YANG library checksum results in a new value for the :yang-library:1.1 capability defined in [I-D.ietf-netconf-nmda-netconf]. Thus, if such a server implements NETCONF notifications [RFC5277], and the notification "netconf-capability-change" [RFC6470], a "netconf-capability-change" notification is generated whenever the YANG library checksum changes.

4. YANG Library YANG Module

The "ietf-yang-library" YANG module imports definitions from "ietf-yang-types" and "ietf-inet-types" defined in [RFC6991] and from "ietf-datastores" defined in [RFC8342]. While the YANG module is defined using YANG version 1.1, the YANG library supports the YANG modules written in any version of YANG.

RFC Ed.: update the date below with the date of RFC publication and remove this note.
<CODE BEGINS> file "ietf-yang-library@2018-02-21.yang"

module ietf-yang-library {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-library";
  prefix "yanglib";

  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-datastores {
    prefix ds;
    // RFC Ed.: update the reference below with the actual RFC number
    reference "RFC XXXX: Network Management Datastore Architecture.";
  }

  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: Juergen Schoenwaelder
      <mailto:j.schoenwaelder@jacobs-university.de>
    Author: Kent Watsen
      <mailto:kwatsen@juniper.net>
    Author: Rob Wilton
      <rwilton@cisco.com>"

  description
    "This module provides information about the YANG modules, datastores, and datastore schemas used by a network management server."
typedef revision-identifier {
  type string {
    pattern '\d{4}-\d{2}-\d{2}';
  }
  description
    "Represents a specific date in YYYY-MM-DD format.";
}

grouping module-identification-leafs {
  ...
description
"Parameters for identifying YANG modules and submodules."

leaf name {
  type yang:yang-identifier;
  mandatory true;
  description
  "The YANG module or submodule name.";
}

leaf revision {
  type revision-identifier;
  description
  "The YANG module or submodule revision date. If no revision
  statement is present in the YANG module or submodule, this
  leaf is not instantiated.";
}

grouping location-leaf-list {
  description
  "Common location leaf list parameter for modules and
  submodules.";

  leaf-list location {
    type inet:uri;
    description
    "Contains a URL that represents the YANG schema
    resource for this module or submodule.

    This leaf will only be present if there is a URL
    available for retrieval of the schema for this entry.";
  }
}

grouping implementation-parameters {
  description
  "Parameters for describing the implementation of a module.";

  leaf-list feature {
    type yang:yang-identifier;
    description
    "List of all YANG feature names from this module that are
    supported by the server, regardless whether they are defined
    in the module or any included submodule.";
  }

  leaf-list deviation {
    type leafref {
      path "../../module/name";
    }
  }
}
grouping module-set-parameters {
  description
  "A set of parameters that describe a module set.";

  leaf name {
    type string;
    description
    "An arbitrary name of the module set.";
  }

  list module {
    key "name";
    description
    "An entry in this list represents a module implemented by the
    server, as per RFC 7950 section 5.6.5, with a particular set
    of supported features and deviations.";
    reference
    "RFC 7950: The YANG 1.1 Data Modeling Language.";
    uses module-identification-leafs;

    leaf namespace {
      type inet:uri;
      mandatory true;
      description
      "The XML namespace identifier for this module.";
    }

    uses location-leaf-list;

    list submodule {
      key "name";
    }
  }
}

Each entry represents one submodule within the parent module.

uses module-identification-leafs;
uses location-leaf-list;
}

uses implementation-parameters;
}
list import-only-module {
  key "name revision";
  description
  "An entry in this list indicates that the server imports reusable definitions from the specified revision of the module, but does not implement any protocol accessible objects from this revision."
  
  Multiple entries for the same module name MAY exist. This can occur if multiple modules import the same module, but specify different revision-dates in the import statements.";

  leaf name {
    type yang:yang-identifier;
    description
    "The YANG module name.";
  }

  leaf revision {
    type union {
      type revision-identifier;
      type string {
        length 0;
      }
    }
    description
    "The YANG module revision date. A zero-length string is used if no revision statement is present in the YANG module.";
  }

  leaf namespace {
    type inet:uri;
    mandatory true;
    description
    "The XML namespace identifier for this module.";
  }
}

uses location-leaf-list;
list submodule {

key "name";
description
  "Each entry represents one submodule within the parent module.";

  uses module-identification-leafs;
  uses location-leaf-list;
}
}

grouping yang-library-parameters {
  description
  "The YANG library data structure is represented as a grouping so it can be reused in configuration or another monitoring data structure.";

  list module-set {
    key name;
    description
      "A set of modules that may be used by one or more schemas. A module set does not have to be referentially complete, i.e., it may define modules that contain import statements for other modules not included in the module set.";

    uses module-set-parameters;
  }

  list schema {
    key "name";
    description
      "A datastore schema that may be used by one or more datastores. The schema must be valid and referentially complete, i.e., it must contain modules to satisfy all used import statements for all modules specified in the schema.";

    leaf name {
      type string;
      description
        "An arbitrary name of the schema.";
    }

    leaf-list module-set {
      type leafref {
        path "../../module-set/name";
      }
    }
  }
}
description
"A set of module-sets that are included in this schema. If a non import-only module appears in multiple module sets, then the module revision and the associated features and deviations must be identical."
}
}

list datastore {
  key "name";
  description
  "A datastore supported by this server.
   Each datastore indicates which schema it supports. The server MUST instantiate one entry in this list per specific datastore it supports.
   Each datastore entry with the same datastore schema SHOULD reference the same schema."
  }
  leaf name {
    type ds:datastore-ref;
    description
    "The identity of the datastore."
  }
  leaf schema {
    type leafref {
      path ".../../schema/name";
    }
    mandatory true;
    description
    "A reference to the schema supported by this datastore. All non import-only modules of the schema are implemented with their associated features and deviations."
  }
}

/*
 * Top-level container
 */

carrier yang-library {
  config false;
  description
  "Container holding the entire YANG library of this server."
}
uses yang-library-parameters;

leaf checksum {
  type string;
  mandatory true;
  description
  "A server-generated checksum of the contents of the 'yang-library' tree. The server MUST change the value of this leaf if the information represented by the 'yang-library' tree, except 'yang-library/checksum', has changed."
}

/* Notifications */

notification yang-library-update {
  description
  "Generated when any YANG library information on the server has changed."

  leaf checksum {
    type leafref {
      path "/yanglib:yang-library/yanglib:checksum";
    }
    mandatory true;
    description
    "Contains the YANG library checksum for the updated YANG library at the time the notification is generated."
  }
}

/* Legacy groupings */

grouping module-list {
  status deprecated;
  description
  "The module data structure is represented as a grouping so it can be reused in configuration or another monitoring data structure."

  grouping common-leafs {
    status deprecated;
    description
  }
}
"Common parameters for YANG modules and submodules."

leaf name {
  type yang:yang-identifier;
  status deprecated;
  description
    "The YANG module or submodule name.";
}

leaf revision {
  type union {
    type revision-identifier;
    type string {
      length 0;
    }
  }
  status deprecated;
  description
    "The YANG module or submodule revision date.
    A zero-length string is used if no revision statement
    is present in the YANG module or submodule.";
}

grouping schema-leaf {
  status deprecated;
  description
    "Common schema leaf parameter for modules and submodules.";
  leaf schema {
    type inet:uri;
    description
      "Contains a URL that represents the YANG schema
      resource for this module or submodule.

      This leaf will only be present if there is a URL
      available for retrieval of the schema for this entry.";
  }
}

list module {
  key "name revision";
  status deprecated;
  description
    "Each entry represents one revision of one module
    currently supported by the server.";

  uses common-leafs {
    status deprecated;
  }
  uses schema-leaf {

status deprecated;
}

leaf namespace {

type inet:uri;

mandatory true;

status deprecated;

description

"The XML namespace identifier for this module."
;
}

leaf-list feature {

type yang:yang-identifier;

status deprecated;

description

"List of YANG feature names from this module that are

supported by the server, regardless whether they are

defined in the module or any included submodule."
;
}

list deviation {

key "name revision";

status deprecated;

description

"List of YANG deviation module names and revisions

used by this server to modify the conformance of

the module associated with this entry. Note that

the same module can be used for deviations for

multiple modules, so the same entry MAY appear

within multiple ‘module’ entries.


The deviation module MUST be present in the ‘module’

list, with the same name and revision values.

The ‘conformance-type’ value will be ‘implement’ for

the deviation module."

uses common-leafs {

status deprecated;

}

}

leaf conformance-type {

type enumeration {

enum implement {

description

"Indicates that the server implements one or more

protocol-accessible objects defined in the YANG module

identified in this entry. This includes deviation

statements defined in the module.


For YANG version 1.1 modules, there is at most one

module entry with conformance type ‘implement’ for a
particular module name, since YANG 1.1 requires that
at most one revision of a module is implemented.

For YANG version 1 modules, there SHOULD NOT be more
than one module entry for a particular module name.";
}
}

enum import {
  description
  "Indicates that the server imports reusable definitions
  from the specified revision of the module, but does
  not implement any protocol accessible objects from
  this revision.

  Multiple module entries for the same module name MAY
  exist. This can occur if multiple modules import the
  same module, but specify different revision-dates in
  the import statements.";
}

mandatory true;
status deprecated;
description
  "Indicates the type of conformance the server is claiming
  for the YANG module identified by this entry.";
}

list submodule {
  key "name revision";
  status deprecated;
description
  "Each entry represents one submodule within the
  parent module.";
  uses common-leafs {
    status deprecated;
  }
  uses schema-leaf {
    status deprecated;
  }
}

/*
 * Legacy operational state data nodes
 */

container modules-state {
  config false;
  status deprecated;
leaf module-set-id {
  type string;
  mandatory true;
  status deprecated;
  description
  "Contains a server-specific identifier representing the current set of modules and submodules. The server MUST change the value of this leaf if the information represented by the 'module' list instances has changed."
}

uses module-list {
  status deprecated;
}

<CODE ENDS>
5. IANA Considerations

RFC 7895 previously registered one URI in the IETF XML registry [RFC3688]. This document takes over this registration entry made by RFC 7895 and changes the Registrant to the IESG according to Section 4 in [RFC3688].


Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

RFC 7895 previously registered one YANG module in the "YANG Module Names" registry [RFC6020] as follows:

   name: ietf-yang-library
   prefix: yanglib
   reference: RFC 7895

This document takes over this registration entry made by RFC 7895.

6. Security Considerations

The YANG module specified in this document defines a schema for data that is accessed by network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

The "/yang-library" subtree of the YANG library may help an attacker identify the server capabilities and server implementations with known bugs since the set of YANG modules supported by a server may reveal the kind of device and the manufacturer of the device.
Although some of this information may be available to all NETCONF users via the NETCONF <hello> message (or similar messages in other management protocols), this YANG module potentially exposes additional details that could be of some assistance to an attacker. Server vulnerabilities may be specific to particular modules, module revisions, module features, or even module deviations. For example, if a particular operation on a particular data node is known to cause a server to crash or significantly degrade device performance, then the module list information will help an attacker identify server implementations with such a defect, in order to launch a denial-of-service attack on the device.

7. Acknowledgments

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).

8. References

8.1. Normative References


Bierman, et al. Expires October 11, 2018
8.2. Informative References

[I-D.ietf-netconf-nmda-netconf]

[I-D.ietf-netconf-nmda-restconf]

[I-D.ietf-netmod-schema-mount]
Appendix A. Summary of Changes from RFC 7895

This document updates [RFC7895] in the following ways:

- Renamed document title from "YANG Module Library" to "YANG Library".
Added a new top-level "/yang-library" container to hold the entire YANG library providing information about module sets, schemas, and datastores.

Refactored the "/modules-state" container into a new "/yang-library/module-set" list.

Added a new "/yang-library/schema" list and a new "/yang-library/datastore" list.

Added a set of new groupings as replacements for the deprecated groupings.

Added a "yang-library-update" notification as a replacement for the deprecated "yang-library-change" notification.

Deprecated the "/modules-state" tree.

Deprecated the "/module-list" grouping.

Deprecated the "/yang-library-change" notification.

Appendix B. Example YANG Library Instance for a Basic Server

The following example shows the YANG Library of a basic server implementing the "ietf-interfaces" [RFC8343] and "ietf-ip" [RFC8344] modules in the <running>, <startup>, and <operational> datastores and the "ietf-hardware" [RFC8348] module in the <operational> datastore.

Newlines in leaf values are added for formatting reasons.

```xml
<yang-library
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library"

    <module-set>
        <name>config-modules</name>
        <module>
            <name>ietf-interfaces</name>
            <revision>2018-01-09</revision> <!-- RFC Ed. update this -->
            <namespace>
                urn:ietf:params:xml:ns:yang:ietf-interfaces
            </namespace>
        </module>
        <module>
            <name>ietf-ip</name>
            <revision>2018-01-09</revision> <!-- RFC Ed. update this -->
            <namespace>

xmlns:yang:ietf-ip
</namespace>
</module>
<import-only-module>
  <name>ietf-yang-types</name>
  <revision>2013-07-15</revision>
  <namespace>
    urn:ietf:params:xml:ns:yang:ietf-yang-types
  </namespace>
</import-only-module>
</import-only-module>
<import-only-module>
  <name>ietf-inet-types</name>
  <revision>2013-07-15</revision>
  <namespace>
    urn:ietf:params:xml:ns:yang:ietf-inet-types
  </namespace>
</import-only-module>
</import-only-module>
</module-set>

<module-set>
  <name>state-modules</name>
  <module>
    <name>ietf-hardware</name>
    <revision>2018-12-18</revision> <!-- RFC Ed. update this -->
    <namespace>
      urn:ietf:params:xml:ns:yang:ietf-hardware
    </namespace>
  </module>
  <import-only-module>
    <name>ietf-inet-types</name>
    <revision>2013-07-15</revision>
    <namespace>
      urn:ietf:params:xml:ns:yang:ietf-inet-types
    </namespace>
  </import-only-module>
  <import-only-module>
    <name>ietf-yang-types</name>
    <revision>2013-07-15</revision>
    <namespace>
      urn:ietf:params:xml:ns:yang:ietf-yang-types
    </namespace>
  </import-only-module>
  <import-only-module>
    <name>iana-hardware</name>
    <revision>2017-12-18</revision> <!-- RFC Ed. update this -->
    <namespace>
      urn:ietf:params:xml:ns:yang:iana-hardware
    </namespace>
  </import-only-module>
</module-set>
Appendix C.  Example YANG Library Instance for an Advanced Server

The following example extends the preceding Basic Server YANG Library example, by using modules from [RFC8345] and [RFC8349], to illustrate a slightly more advanced server that:

- Has a module with features only enabled in <operational>; the "ietf-routing module" is supported in <running>, <startup>, and <operational>, but the "multiple-ribs" and "router-id" features are only enabled in <operational>. Hence the "router-id" leaf may be read but not configured.

- Supports a dynamic configuration datastore "example-ds-ephemeral", with only the "ietf-network" and "ietf-network-topology" modules configurable via a notional dynamic configuration protocol.

- Shows an example of datastore specific deviations. The module "example-vendor-hardware-deviations" is included in the schema for
<yang-library>
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library"
    xmlns:ex-ds-eph="urn:example:ds-ephemeral">

    <module-set>
        <name>config-state-modules</name>
        <module>
            <name>ietf-interfaces</name>
            <revision>2018-01-09</revision> <!-- RFC Ed. update this -->
            <namespace>
                urn:ietf:params:xml:ns:yang:ietf-interfaces
            </namespace>
        </module>
        <module>
            <name>ietf-ip</name>
            <revision>2018-01-09</revision> <!-- RFC Ed. update this -->
            <namespace>
            </namespace>
        </module>
        <module>
            <name>ietf-routing</name>
            <revision>2018-01-25</revision> <!-- RFC Ed. update this -->
            <namespace>
            </namespace>
        </module>
        <import-only-module>
            <name>ietf-yang-types</name>
            <revision>2013-07-15</revision>
            <namespace>
                urn:ietf:params:xml:ns:yang:ietf-yang-types
            </namespace>
        </import-only-module>
        <import-only-module>
            <name>ietf-inet-types</name>
            <revision>2013-07-15</revision>
            <namespace>
                urn:ietf:params:xml:ns:yang:ietf-inet-types
            </namespace>
        </import-only-module>
    </module-set>
</yang-library>
</module-set>

<module-set>
  <name>config-only-modules</name>
  <module>
    <name>ietf-routing</name>
    <revision>2018-01-25</revision> <!-- RFC Ed. update this -->
    <namespace>
    </namespace>
  </module>
</module-set>

<module-set>
  <name>dynamic-config-state-modules</name>
  <module>
    <name>ietf-network</name>
    <revision>2017-12-18</revision> <!-- RFC Ed. update this -->
    <namespace>
      urn:ietf:params:xml:ns:yang:ietf-network
    </namespace>
  </module>
  <module>
    <name>ietf-network-topology</name>
    <revision>2017-12-18</revision> <!-- RFC Ed. update this -->
    <namespace>
    </namespace>
  </module>
  <import-only-module>
    <name>ietf-inet-types</name>
    <revision>2013-07-15</revision>
    <namespace>
      urn:ietf:params:xml:ns:yang:ietf-inet-types
    </namespace>
  </import-only-module>
</module-set>

<module-set>
  <name>state-only-modules</name>
  <module>
    <name>ietf-hardware</name>
    <revision>2018-12-18</revision> <!-- RFC Ed. update this -->
    <namespace>
      urn:ietf:params:xml:ns:yang:ietf-hardware
    </namespace>
    <deviation>example-vendor-hardware-deviations</deviation>
  </module>
</module-set>
<module>
  <name>ietf-routing</name>
  <revision>2018-01-25</revision> <!-- RFC Ed. update this -->
  <namespace>
  </namespace>
  <feature>multiple-ribs</feature>
  <feature>router-id</feature>
</module>

<module>
  <name>example-vendor-hardware-deviations</name>
  <revision>2018-01-31</revision>
  <namespace>
    urn:example:example-vendor-hardware-deviations
  </namespace>
</module>

<import-only-module>
  <name>ietf-inet-types</name>
  <revision>2013-07-15</revision>
  <namespace>
    urn:ietf:params:xml:ns:yang:ietf-inet-types
  </namespace>
</import-only-module>

<import-only-module>
  <name>ietf-yang-types</name>
  <revision>2013-07-15</revision>
  <namespace>
    urn:ietf:params:xml:ns:yang:ietf-yang-types
  </namespace>
</import-only-module>

<import-only-module>
  <name>iana-hardware</name>
  <revision>2017-12-18</revision> <!-- RFC Ed. update this -->
  <namespace>
    urn:ietf:params:xml:ns:yang:iana-hardware
  </namespace>
</import-only-module>

</module-set>

<schema>
  <name>config-schema</name>
  <module-set>config-state-modules</module-set>
  <module-set>config-only-modules</module-set>
</schema>

<schema>
  <name>dynamic-config-schema</name>
  <module-set>dynamic-config-state-modules</module-set>
</schema>
<schema>
  <name>state-schema</name>
  <module-set>config-state-modules</module-set>
  <module-set>dynamic-config-state-modules</module-set>
  <module-set>state-only-modules</module-set>
</schema>

<datastore>
  <name>ds:startup</name>
  <schema>config-schema</schema>
</datastore>

<datastore>
  <name>ds:running</name>
  <schema>config-schema</schema>
</datastore>

<datastore>
  <name>ex-ds-eph:ds-ephemeral</name>
  <schema>dynamic-config-schema</schema>
</datastore>

<datastore>
  <name>ds:operational</name>
  <schema>state-schema</schema>
</datastore>

<checksum>14782ab9bd56b92aacc156a2958fbe12312fb285</checksum>
</yang-library>

Authors’ Addresses

Andy Bierman
YumaWorks
Email: andy@yumaworks.com

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

Juergen Schoenwaelder
Jacobs University
Email: j.schoenwaelder@jacobs-university.de
Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net

Robert Wilton
Cisco Systems

Email: rwilton@cisco.com
Abstract

This document defines three YANG modules: the first defines groupings for a generic SSH client, the second defines groupings for a generic SSH server, and the third defines common identities and groupings used by both the client and the server. It is intended that these groupings will be used by applications using the SSH protocol.

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.

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

- I-D.ietf-netconf-trust-anchors
- I-D.ietf-netconf-keystore

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

- "XXXX" --> the assigned RFC value for this draft
- "YYYY" --> the assigned RFC value for I-D.ietf-netconf-trust-anchors
- "ZZZZ" --> the assigned RFC value for I-D.ietf-netconf-keystore

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

- "2018-06-04" --> the publication date of this draft
The following Appendix section is to be removed prior to publication:

- Appendix A. 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 December 6, 2018.

Copyright Notice

Copyright (c) 2018 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 ............................................. 3
2. Terminology ............................................. 4
3. The SSH Client Model ..................................... 4
   3.1. Tree Diagram ....................................... 4
   3.2. Example Usage ..................................... 6
   3.3. YANG Module ....................................... 9
4. The SSH Server Model .................................... 12
   4.1. Tree Diagram ..................................... 12
   4.2. Example Usage ..................................... 14
   4.3. YANG Module ..................................... 17
This document defines three YANG 1.1 [RFC7950] modules: the first defines a grouping for a generic SSH client, the second defines a grouping for a generic SSH server, and the third defines identities and groupings common to both the client and the server. It is intended that these groupings will be used by applications using the SSH protocol [RFC4252], [RFC4253], and [RFC4254]. For instance, these groupings could be used to help define the data model for an OpenSSH [OPENSSH] server or a NETCONF over SSH [RFC6242] based server.

The client and server YANG modules in this document each define one grouping, which is focused on just SSH-specific configuration, and specifically avoids any transport-level configuration, such as what ports to listen on or connect to. This affords applications the opportunity to define their own strategy for how the underlying TCP connection is established. For instance, applications supporting NETCONF Call Home [RFC8071] could use the "ssh-server-grouping" grouping for the SSH parts it provides, while adding data nodes for the TCP-level call-home configuration.

The modules defined in this document uses groupings defined in [I-D.ietf-netconf-keystore] enabling keys to be either locally defined or a reference to globally configured values.
The modules defined in this document optionally support [RFC6187] enabling X.509v3 certificate based host keys and public keys.

2. 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. The SSH Client Model

3.1. Tree Diagram

This section provides two tree diagrams [RFC8340] for the "ietf-ssh-client" module, the first with used groupings expanded and the second with used groupings not expanded.

The following tree diagram has used groupings expanded:
module: ietf-ssh-client

grouping ssh-client-grouping
  +-- client-identity
  |  +-- username?            string
  |  +-- (auth-type)
  |     +--:(password)
  |     |  +-- password?      string
  |  +--:(public-key)
  |     +-- public-key
  |     |  +-- (local-or-keystore)
  |     |     +--:(local)
  |     |     |  +-- algorithm ct:key-algorithm-ref
  |     |     |  +-- public-key binary
  |     |     |  +-- private-key union
  |     +--:(keystore) (keystore-implemented)?
  |          +-- reference ks:asymmetric-key-ref
  |  +--:(certificate)
  |     +-- certificate (sshcmn:ssh-x509-certs)?
  |     +-- (local-or-keystore)
  |          +--:(local)
  |          |  +-- algorithm ct:key-algorithm-ref
  |          |  +-- public-key binary
  |          |  +-- private-key union
  |          |  +-- cert
  |          |     +-- ct:end-entity-cert-cms
  |          |     +-- certificate-expiration
  |          |     |  +-- expiration-date? yang:date-and-time
  |          +--:(keystore) (keystore-implemented)?
  |                 +-- reference
  |                 ks:asymmetric-key-certificate-ref
  +-- server-auth
  +-- pinned-ssh-host-keys? ta:pinned-host-keys-ref
  +-- pinned-ca-certs? ta:pinned-certificates-ref
  |     {sshcmn:ssh-x509-certs}?
  +-- pinned-server-certs? ta:pinned-certificates-ref
  |     {sshcmn:ssh-x509-certs}?
  +-- transport-params {ssh-client-transport-params-config}?
  +-- host-key
  |  +-- host-key-alg* identityref
  +-- key-exchange
  |  +-- key-exchange-alg* identityref
  +-- encryption
  |  +-- encryption-alg* identityref
  +-- mac
  |  +-- mac-alg* identityref
The following tree diagram does not have the groupings expanded:

```
module: ietf-ssh-client

grouping ssh-client-grouping
  +-- client-identity
    |  +-- username?            string
    |  +-- (auth-type)
    |     +--:(password)
    |     |  +-- password?      string
    |     +--:(public-key)
    |     |  +-- u ks:local-or-keystore-asymmetric-key-grouping
    |     +--:(certificate)
    |        +-- certificate {sshcmn:ssh-x509-certs}?
    |        +----u ks:local-or-keystore-end-entity-certificate-grouping
  +-- server-auth
    |  +-- pinned-ssh-host-keys?   ta:pinned-host-keys-ref
    |  +-- pinned-ca-certs?        ta:pinned-certificates-ref
    |       {sshcmn:ssh-x509-certs}?
    |  +-- pinned-server-certs?    ta:pinned-certificates-ref
    |       {sshcmn:ssh-x509-certs}?
    |  +-- transport-params {ssh-client-transport-params-config}?
    |       +----u sshcmn:transport-params-grouping
```

### 3.2. Example Usage

This section presents two examples showing the ssh-client-grouping populated with some data. These examples are effectively the same except the first configures the client identity using a local key while the second uses a key configured in a keystore. Both examples are consistent with the examples presented in Section 3 of [I-D.ietf-netconf-trust-anchors] and Section 3.2 of [I-D.ietf-netconf-keystore].

The following example configures the client identity using a local key:
<ssh-client
    xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-client"

<!-- how this client will authenticate itself to the server -->
<client-identity>
    <username>foobar</username>
    <public-key>
        <private-key>base64encodedvalue==</private-key>
        <public-key>base64encodedvalue==</public-key>
    </public-key>
</client-identity>

<!-- which host-keys will this client trust -->
<server-auth>
    <pinned-ssh-host-keys>explicitly-trusted-ssh-host-keys</pinned-ssh-host-keys>
</server-auth>

<transport-params>
    <host-key>
        <host-key-alg>algs:ssh-rsa</host-key-alg>
    </host-key>
    <key-exchange>
        <key-exchange-alg>
            algs:diffie-hellman-group-exchange-sha256
        </key-exchange-alg>
    </key-exchange>
    <encryption>
        <encryption-alg>algs:aes256-ctr</encryption-alg>
        <encryption-alg>algs:aes192-ctr</encryption-alg>
        <encryption-alg>algs:aes128-ctr</encryption-alg>
        <encryption-alg>algs:aes256-cbc</encryption-alg>
        <encryption-alg>algs:aes192-cbc</encryption-alg>
        <encryption-alg>algs:aes128-cbc</encryption-alg>
    </encryption>
    <mac>
        <mac-alg>algs:hmac-sha2-256</mac-alg>
        <mac-alg>algs:hmac-sha2-512</mac-alg>
    </mac>
</transport-params>
</ssh-client>
The following example configures the client identity using a key from the keystore:

[Note: '\ line wrapping for formatting only]

```xml
<ssh-client
  xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-client"

  <!-- how this client will authenticate itself to the server -->
  <client-identity>
    <username>foobar</username>
    <public-key>
      <reference>ex-rsa-key</reference>
    </public-key>
  </client-identity>

  <!-- which host-keys will this client trust -->
  <server-auth>
    <pinned-ssh-host-keys>explicitly-trusted-ssh-host-keys</pinned-ssh-host-keys>
  </server-auth>

  <transport-params>
    <host-key>
      <host-key-alg>algs:ssh-rsa</host-key-alg>
    </host-key>
    <key-exchange>
      <key-exchange-alg>
        algs:diffie-hellman-group-exchange-sha256
      </key-exchange-alg>
    </key-exchange>
    <encryption>
      <encryption-alg>algs:aes256-ctr</encryption-alg>
      <encryption-alg>algs:aes192-ctr</encryption-alg>
      <encryption-alg>algs:aes128-ctr</encryption-alg>
      <encryption-alg>algs:aes256-cbc</encryption-alg>
      <encryption-alg>algs:aes192-cbc</encryption-alg>
      <encryption-alg>algs:aes128-cbc</encryption-alg>
    </encryption>
    <mac>
      <mac-alg>algs:hmac-sha2-256</mac-alg>
      <mac-alg>algs:hmac-sha2-512</mac-alg>
    </mac>
  </transport-params>

</ssh-client>
```
3.3. YANG Module

This YANG module has normative references to [RFC6536],
[I-D.ietf-netconf-trust-anchors], and [I-D.ietf-netconf-keystore].

<CODE BEGINS> file "ietf-ssh-client@2018-06-04.yang"
module ietf-ssh-client {
  yang-version 1.1;
  prefix "sshc";

  import ietf-ssh-common {
    prefix sshcmn;
    revision-date 2018-06-04; // stable grouping definitions
    reference
      "RFC XXXX: YANG Groupings for SSH Clients and SSH Servers";
  }

  import ietf-trust-anchors {
    prefix ta;
    reference
      "RFC YYYY: YANG Data Model for Global Trust Anchors";
  }

  import ietf-keystore {
    prefix ks;
    reference
      "RFC ZZZZ: YANG Data Model for a Centralized Keystore Mechanism";
  }

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

  contact
    "WG Web: <http://datatracker.ietf.org/wg/netconf/>
    WG List: <mailto:netconf@ietf.org>
    Author: Kent Watsen
    <mailto:kwatsen@juniper.net>
    Author: Gary Wu
    <mailto:garywu@cisco.com>";

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

Copyright (c) 2018 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 "2018-06-04" {
  description
    "Initial version";
  reference
    "RFC XXXX: YANG Groupings for SSH Clients and SSH Servers";
}

// features

feature ssh-client-transport-params-config {
  description
    "SSH transport layer parameters are configurable on an SSH
    client.";
}

// groupings

grouping ssh-client-grouping {
  description
    "A reusable grouping for configuring a SSH client without
    any consideration for how an underlying TCP session is
    established.";

grouping ssh-client-grouping {
  description
    "A reusable grouping for configuring a SSH client without
    any consideration for how an underlying TCP session is
    established.";

ccontainer client-identity {
  description
    "The credentials used by the client to authenticate to
    the SSH server.";

  leaf username {
    type string;
    description
      "The username of this user. This will be the username
      used, for instance, to log into an SSH server.";
  }
}
choice auth-type {
    mandatory true;
    description 'The authentication type.';
    leaf password {
        type string;
        description 'A password to be used for client authentication.';
    }
}

container public-key {
    uses ks:local-or-keystore-asymmetric-key-grouping;
    description 'A locally-defined or referenced asymmetric key pair to be used for client authentication.';
    reference "RFC ZZZZ: YANG Data Model for a Centralized Keystore Mechanism";
}

container certificate {
    if-feature sshcmn:ssh-x509-certs;
    uses ks:local-or-keystore-end-entity-certificate-grouping;
    description 'A locally-defined or referenced certificate to be used for client authentication.';
    reference "RFC ZZZZ: YANG Data Model for a Centralized Keystore Mechanism";
}

container server-auth {
    must 'pinned-ssh-host-keys or pinned-ca-certs or ' + 'pinned-server-certs';
    description 'Trusted server identities.';
    leaf pinned-ssh-host-keys {
        type ta:pinned-host-keys-ref;
        description 'A reference to a list of SSH host keys used by the SSH client to authenticate SSH server host keys. A server host key is authenticated if it is an exact match to a configured SSH host key.';
        reference "RFC YYYY: YANG Data Model for Global Trust Anchors";
    }
}

Watsen & Wu Expires December 6, 2018 [Page 11]
leaf pinned-ca-certs {
  if-feature sshcmn:ssh-x509-cert;
  type ta:pinned-certificates-ref;
  description
    "A reference to a list of certificate authority (CA)
certificates used by the SSH client to authenticate
SSH server certificates. A server certificate is
authenticated if it has a valid chain of trust to
a configured CA certificate."
  reference
    "RFC YYYY: YANG Data Model for Global Trust Anchors";
}

leaf pinned-server-certs {
  if-feature sshcmn:ssh-x509-cert;
  type ta:pinned-certificates-ref;
  description
    "A reference to a list of server certificates used by
the SSH client to authenticate SSH server certificates.
A server certificate is authenticated if it is an
exact match to a configured server certificate."
  reference
    "RFC YYYY: YANG Data Model for Global Trust Anchors";
}
} // end server-auth

container transport-params {
  if-feature ssh-client-transport-params-config;
  description
    "Configurable parameters of the SSH transport layer."
  uses sshcmn:transport-params-grouping;
}
} // end ssh-client-grouping

<CODE ENDS>

4. The SSH Server Model

4.1. Tree Diagram

This section provides two tree diagrams [RFC8340] for the "ietf-ssh-server" module, the first with used groupings expanded and the second with used groupings not expanded.

The following tree diagram has used groupings expanded:
module: ietf-ssh-server

grouping ssh-server-grouping
  +-- server-identity
    |  +-- host-key* [name]
    |    +-- name?                string
    |    +-- (host-key-type)
    |    |  +--:(public-key)
    |    |     +-- public-key
    |    |     |  +-- (local-or-keystore)
    |    |     |     +-- algorithm
    |    |     |     +-- public-key
    |    |     |     +-- private-key
    |    |     |     +--:(keystore) {keystore-implemented}?
    |    |     |        +-- reference
    |    |     +--:(certificate)
    |    |     |  +-- certificate {sshcmn:ssh-x509-certs}?
    |    |     |     +-- (local-or-keystore)
    |    |     |     |  +-- algorithm
    |    |     |     |  |       ct:key-algorithm-ref
    |    |     |     |  +-- public-key
    |    |     |     |  +-- private-key
    |    |     |     |  +-- cert
    |    |     |     |    |       ct:end-entity-cert-cms
    |    |     |     |    |       certificate-expiration
    |    |     |     |    |       +-- expiration-date?   yang:date-and-time
    |    |     |    |      +--:(keystore) {keystore-implemented}?
    |    |     |    |         +-- reference
    |    |     |    |         ks:asymmetric-key-certificate-ref
    |    |     +-- client-cert-auth {sshcmn:ssh-x509-certs}?
    |    |     |  +-- pinned-ca-certs?
    |    |     |     +-- pinned-client-certs?
    |    |     +-- transport-params {ssh-server-transport-params-config}?
    |    |        +-- host-key
    |    |        |  +-- host-key-alg*  identityref
    |    |        |  +-- key-exchange
    |    |        |     +-- key-exchange-alg*  identityref
    |    |        |  +-- encryption
    |    |        |     +-- encryption-alg*  identityref
    |    |        |  +-- mac
    |    |        |     +-- mac-alg*  identityref

The following tree diagram does not have the used groupings expanded:
module: ietf-ssh-server

grouping ssh-server-grouping
    +-- server-identity
        |    +-- host-key* [name]
        |        |    +-- name?                string
        |        |    +-- host-key-type
        |        |       +--:(public-key)
        |        |           |    |    +-- public-key
        |        |           |    |       +-- u ks:local-or-keystore-asymmetric-key-grouping
        |        |       +--:(certificate)
        |        |           |    |    +-- u ks:local-or-keystore-end-entity-certificate
        |        +-- client-cert-auth {sshcmn:ssh-x509-certs}?
        |    +--- u sshcmn:transport-params-grouping
        +-- transport-params {ssh-server-transport-params-config}?

4.2. Example Usage

This section presents two examples showing the ssh-server-grouping populated with some data. These examples are effectively the same except the first configures the server identity using a local key while the second uses a key configured in a keystore. Both examples are consistent with the examples presented in Section 3 of [I-D.ietf-netconf-trust-anchors] and Section 3.2 of [I-D.ietf-netconf-keystore].

The following example configures the server identity using a local key:

[Note: ‘\’ line wrapping for formatting only]

<ssh-server xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-server"
    <server-identity>
        <host-key>
            <name>deployment-specific-certificate</name>
            <public-key>

Watsen & Wu             Expires December 6, 2018               [Page 14]
The following example configures the server identity using a key from the keystore:
[Note: ‘\’ line wrapping for formatting only]

<ssh-server xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-server"

  <!-- which host-keys will this SSH server present -->
  <server-identity>
    <host-key>
      <name>deployment-specific-certificate</name>
      <public-key>
        <reference>ex-rsa-key</reference>
      </public-key>
    </host-key>
  </server-identity>

  <!-- which client-certs will this SSH server trust -->
  <client-cert-auth>
    <pinned-ca-certs>explicitly-trusted-client-ca-certs</pinned-ca-certs>
    <pinned-client-certs>explicitly-trusted-client-certs</pinned-client-certs>
  </client-cert-auth>

  <transport-params>
    <host-key>
      <host-key-alg>algs:ssh-rsa</host-key-alg>
    </host-key>
    <key-exchange>
      <key-exchange-alg>
        algs:diffie-hellman-group-exchange-sha256
      </key-exchange-alg>
    </key-exchange>
    <encryption>
      <encryption-alg>algs:aes256-ctr</encryption-alg>
      <encryption-alg>algs:aes192-ctr</encryption-alg>
      <encryption-alg>algs:aes128-ctr</encryption-alg>
      <encryption-alg>algs:aes256-cbc</encryption-alg>
      <encryption-alg>algs:aes192-cbc</encryption-alg>
      <encryption-alg>algs:aes128-cbc</encryption-alg>
    </encryption>
    <mac>
      <mac-alg>algs:hmac-sha2-256</mac-alg>
      <mac-alg>algs:hmac-sha2-512</mac-alg>
    </mac>
  </transport-params>

</ssh-server>
4.3. YANG Module

This YANG module has normative references to [I-D.ietf-netconf-trust-anchors] and [I-D.ietf-netconf-keystore] and informative references to [RFC4253] and [RFC7317].

<CODE BEGINS> file "ietf-ssh-server@2018-06-04.yang"
module ietf-ssh-server {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-ssh-server";
  prefix "sshs";

  import ietf-ssh-common {
    prefix sshcmn;
    revision-date 2018-06-04; /* stable grouping definitions */
    reference "RFC XXXX: YANG Groupings for SSH Clients and SSH Servers";
  }

  import ietf-trust-anchors {
    prefix ta;
    reference "RFC YYY: YANG Data Model for Global Trust Anchors";
  }

  import ietf-keystore {
    prefix ks;
    reference "RFC ZZZZ: YANG Data Model for a Centralized Keystore Mechanism";
  }

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

  contact "WG Web: <http://datatracker.ietf.org/wg/netconf/>"
             "WG List: <mailto:netconf@ietf.org>"
             "Author: Kent Watsen <mailto:kwatsen@juniper.net>"
             "Author: Gary Wu <mailto:garywu@cisco.com>"

  description

Watsen & Wu Expires December 6, 2018 [Page 17]
"This module defines a reusable grouping for a SSH server that can be used as a basis for specific SSH server instances.

Copyright (c) 2018 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 "2018-06-04" {
  description
    "Initial version";
  reference
    "RFC XXXX: YANG Groupings for SSH Clients and SSH Servers";
}

// features

feature ssh-server-transport-params-config {
  description
    "SSH transport layer parameters are configurable on an SSH server.";
}

// groupings

grouping ssh-server-grouping {
  description
    "A reusable grouping for configuring a SSH server without any consideration for how underlying TCP sessions are established.";

container server-identity {
  description
    "The list of host-keys the SSH server will present when establishing a SSH connection.";
  list host-key {
    key name;
    min-elements 1;
    ordered-by user;
    description
}
An ordered list of host keys the SSH server will use to construct its ordered list of algorithms, when sending its SSH_MSG_KEXINIT message, as defined in Section 7.1 of RFC 4253.

reference
"RFC 4253: The Secure Shell (SSH) Transport Layer Protocol"

leaf name {
  type string;
  description
  "An arbitrary name for this host-key";
}

choice host-key-type {
  mandatory true;
  description
  "The type of host key being specified";
  container public-key {
    uses ks:local-or-keystore-asymmetric-key-grouping;
    description
    "A locally-defined or referenced asymmetric key pair to be used for the SSH server’s host key."
    reference
    "RFC ZZZZ: YANG Data Model for a Centralized Keystore Mechanism"
  }
  container certificate {
    if-feature sshcmn:ssh-x509-certs;
    uses
    ks:local-or-keystore-end-entity-certificate-grouping;
    description
    "A locally-defined or referenced end-entity certificate to be used for the SSH server’s host key."
    reference
    "RFC ZZZZ: YANG Data Model for a Centralized Keystore Mechanism"
  }
}

container client-cert-auth {
  if-feature sshcmn:ssh-x509-certs;
  description
  "A reference to a list of pinned certificate authority (CA) certificates and a reference to a list of pinned client certificates."
Note: password and public-key based client authentication is not configured in this YANG module as they are expected to be configured by the ietf-system module defined in RFC 7317.

Reference: "RFC 7317: A YANG Data Model for System Management".

leaf pinned-ca-certs {
  type ta:pinned-certificates-ref;
  description
  "A reference to a list of certificate authority (CA) certificates used by the SSH server to authenticate SSH client certificates. A client certificate is authenticated if it has a valid chain of trust to a configured pinned CA certificate."
  reference
  "RFC YYYY: YANG Data Model for Global Trust Anchors";
}

leaf pinned-client-certs {
  type ta:pinned-certificates-ref;
  description
  "A reference to a list of client certificates used by the SSH server to authenticate SSH client certificates. A clients certificate is authenticated if it is an exact match to a configured pinned client certificate."
  reference
  "RFC YYYY: YANG Data Model for Global Trust Anchors";
}

container transport-params {
  if-feature ssh-server-transport-params-config;
  description
  "Configurable parameters of the SSH transport layer."
  uses sshcmn:transport-params-grouping;
}

} // end ssh-server-grouping

5. The SSH Common Model

The SSH common model presented in this section contains identities and groupings common to both SSH clients and SSH servers. The transport-params-grouping can be used to configure the list of SSH transport algorithms permitted by the SSH client or SSH server. The lists of algorithms are ordered such that, if multiple algorithms are
permitted by the client, the algorithm that appears first in its list that is also permitted by the server is used for the SSH transport layer connection. The ability to restrict the the algorithms allowed is provided in this grouping for SSH clients and SSH servers that are capable of doing so and may serve to make SSH clients and SSH servers compliant with security policies.

Features are defined for algorithms that are OPTIONAL or are not widely supported by popular implementations. Note that the list of algorithms is not exhaustive. As well, some algorithms that are REQUIRED by [RFC4253] are missing, notably "ssh-dss" and "diffie-hellman-group1-sha1" due to their weak security and there being alternatives that are widely supported.

5.1. Tree Diagram

The following tree diagram [RFC8340] provides an overview of the data model for the "ietf-ssh-common" module.

```
module: ietf-ssh-common

grouping transport-params-grouping
    +-- host-key
        | +-- host-key-alg* identityref
        +-- key-exchange
            | +-- key-exchange-alg* identityref
            +-- encryption
                | +-- encryption-alg* identityref
                +-- mac
                    +-- mac-alg* identityref
```

5.2. Example Usage

This following example illustrates how the transport-params-grouping appears when populated with some data.
<transport-params
 xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-common"
 <host-key>
  <host-key-alg>algs:x509v3-rsa2048-sha256</host-key-alg>
  <host-key-alg>algs:ssh-rsa</host-key-alg>
 </host-key>
 <key-exchange>
  <key-exchange-alg>
   algs:diffie-hellman-group-exchange-sha256
  </key-exchange-alg>
 </key-exchange>
 <key-exchange>
  <key-exchange-alg>
   algs:diffie-hellman-group-exchange-sha256
  </key-exchange-alg>
 </key-exchange>
 <encryption>
  <encryption-alg>algs:aes256-ctr</encryption-alg>
  <encryption-alg>algs:aes192-ctr</encryption-alg>
  <encryption-alg>algs:aes128-ctr</encryption-alg>
  <encryption-alg>algs:aes256-cbc</encryption-alg>
  <encryption-alg>algs:aes192-cbc</encryption-alg>
  <encryption-alg>algs:aes128-cbc</encryption-alg>
 </encryption>
 <mac>
  <mac-alg>algs:hmac-sha2-256</mac-alg>
  <mac-alg>algs:hmac-sha2-512</mac-alg>
 </mac>
</transport-params>

5.3. YANG Module

This YANG module has normative references to [RFC4253], [RFC4344],
[RFC4419], [RFC5656], [RFC6187], and [RFC6668].

<CODE BEGINS> file "ietf-ssh-common@2018-06-04.yang"
module ietf-ssh-common {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-ssh-common";
  prefix "sshcmn";

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

  contact
   "WG Web: <http://datatracker.ietf.org/wg/netconf/>
   WG List: <mailto:netconf@ietf.org>
   Author: Kent Watsen
   <mailto:kwatsen@juniper.net>

Watsen & Wu Expires December 6, 2018 [Page 22]
description
"This module defines a common features, identities, and groupings for Secure Shell (SSH).

Copyright (c) 2018 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 "2018-06-04" {
  description
    "Initial version";
  reference
    "RFC XXXX: YANG Groupings for SSH Clients and SSH Servers";
}

// features

feature ssh-ecc {
  description
    "Elliptic Curve Cryptography is supported for SSH.";
  reference
    "RFC 5656: Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer";
}

feature ssh-x509-certs {
  description
    "X.509v3 certificates are supported for SSH per RFC 6187.";
  reference
    "RFC 6187: X.509v3 Certificates for Secure Shell Authentication";
}

feature ssh-dh-group-exchange {
  description
    "Diffie-Hellman Group Exchange is supported for SSH.";
feature ssh-ctr {
  description
    "SDCTR encryption mode is supported for SSH.";
  reference
    "RFC 4344: The Secure Shell (SSH) Transport Layer Encryption Modes";
}

feature ssh-sha2 {
  description
    "The SHA2 family of cryptographic hash functions is supported for SSH.";
  reference
    "FIPS PUB 180-4: Secure Hash Standard (SHS)";
}

// identities

identity public-key-alg-base {
  description
    "Base identity used to identify public key algorithms.";
}

identity ssh-dss {
  base public-key-alg-base;
  description
    "Digital Signature Algorithm using SHA-1 as the hashing algorithm.";
  reference
    "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
}

identity ssh-rsa {
  base public-key-alg-base;
  description
    "RSASSA-PKCS1-v1_5 signature scheme using SHA-1 as the hashing algorithm.";
  reference
    "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
}
identity ecdsa-sha2-nistp256 {
    base public-key-alg-base;
    if-feature "ssh-ecc and ssh-sha2";
    description
        "Elliptic Curve Digital Signature Algorithm (ECDSA) using the
        nistp256 curve and the SHA2 family of hashing algorithms."
    reference
        "RFC 5656: Elliptic Curve Algorithm Integration in the
        Secure Shell Transport Layer";
}

identity ecdsa-sha2-nistp384 {
    base public-key-alg-base;
    if-feature "ssh-ecc and ssh-sha2";
    description
        "Elliptic Curve Digital Signature Algorithm (ECDSA) using the
        nistp384 curve and the SHA2 family of hashing algorithms."
    reference
        "RFC 5656: Elliptic Curve Algorithm Integration in the
        Secure Shell Transport Layer";
}

identity ecdsa-sha2-nistp521 {
    base public-key-alg-base;
    if-feature "ssh-ecc and ssh-sha2";
    description
        "Elliptic Curve Digital Signature Algorithm (ECDSA) using the
        nistp521 curve and the SHA2 family of hashing algorithms."
    reference
        "RFC 5656: Elliptic Curve Algorithm Integration in the
        Secure Shell Transport Layer";
}

identity x509v3-ssh-rsa {
    base public-key-alg-base;
    if-feature ssh-x509-certs;
    description
        "RSASSA-PKCS1-v1_5 signature scheme using a public key stored
        in an X.509v3 certificate and using SHA-1 as the hashing
        algorithm."
    reference
        "RFC 6187: X.509v3 Certificates for Secure Shell
        Authentication";
}

identity x509v3-rsa2048-sha256 {
    base public-key-alg-base;
    if-feature "ssh-x509-certs and ssh-sha2";
description
"RSASSA-PKCS1-v1_5 signature scheme using a public key stored
in an X.509v3 certificate and using SHA-256 as the hashing
algorithm. RSA keys conveyed using this format MUST have a
modulus of at least 2048 bits."
reference
"RFC 6187: X.509v3 Certificates for Secure Shell
Authentication";

identity x509v3-ecdsa-sha2-nistp256 {
  base public-key-alg-base;
  if-feature "ssh-ecc and ssh-x509-certs and ssh-sha2";
  description
  "Elliptic Curve Digital Signature Algorithm (ECDSA)
  using the nistp256 curve with a public key stored in
  an X.509v3 certificate and using the SHA2 family of
  hashing algorithms.";
  reference
  "RFC 6187: X.509v3 Certificates for Secure Shell
  Authentication";
}

identity x509v3-ecdsa-sha2-nistp384 {
  base public-key-alg-base;
  if-feature "ssh-ecc and ssh-x509-certs and ssh-sha2";
  description
  "Elliptic Curve Digital Signature Algorithm (ECDSA)
  using the nistp384 curve with a public key stored in
  an X.509v3 certificate and using the SHA2 family of
  hashing algorithms.";
  reference
  "RFC 6187: X.509v3 Certificates for Secure Shell
  Authentication";
}

identity x509v3-ecdsa-sha2-nistp521 {
  base public-key-alg-base;
  if-feature "ssh-ecc and ssh-x509-certs and ssh-sha2";
  description
  "Elliptic Curve Digital Signature Algorithm (ECDSA)
  using the nistp521 curve with a public key stored in
  an X.509v3 certificate and using the SHA2 family of
  hashing algorithms.";
  reference
  "RFC 6187: X.509v3 Certificates for Secure Shell
  Authentication";
}
identity key-exchange-alg-base {
    description
    "Base identity used to identify key exchange algorithms.";
}

identity diffie-hellman-group14-sha1 {
    base key-exchange-alg-base;
    description
    "Diffie-Hellman key exchange with SHA-1 as HASH and
    Oakley Group 14 (2048-bit MODP Group).";
    reference
    "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
}

identity diffie-hellman-group-exchange-sha1 {
    base key-exchange-alg-base;
    if-feature ssh-dh-group-exchange;
    description
    "Diffie-Hellman Group and Key Exchange with SHA-1 as HASH.";
    reference
    "RFC 4419: Diffie-Hellman Group Exchange for the
    Secure Shell (SSH) Transport Layer Protocol";
}

identity diffie-hellman-group-exchange-sha256 {
    base key-exchange-alg-base;
    if-feature "ssh-dh-group-exchange and ssh-sha2";
    description
    "Diffie-Hellman Group and Key Exchange with SHA-256 as HASH.";
    reference
    "RFC 4419: Diffie-Hellman Group Exchange for the
    Secure Shell (SSH) Transport Layer Protocol";
}

identity ecdh-sha2-nistp256 {
    base key-exchange-alg-base;
    if-feature "ssh-ecc and ssh-sha2";
    description
    "Elliptic Curve Diffie-Hellman (ECDH) key exchange using the
    nistp256 curve and the SHA2 family of hashing algorithms.";
    reference
    "RFC 5656: Elliptic Curve Algorithm Integration in the
    Secure Shell Transport Layer";
}

identity ecdh-sha2-nistp384 {
    base key-exchange-alg-base;
    if-feature "ssh-ecc and ssh-sha2";
description
   "Elliptic Curve Diffie-Hellman (ECDH) key exchange using the
   nistp384 curve and the SHA2 family of hashing algorithms.";
reference
   "RFC 5656: Elliptic Curve Algorithm Integration in the
   Secure Shell Transport Layer";
)

identity ecdh-sha2-nistp521 {
   base key-exchange-alg-base;
   if-feature "ssh-ecc and ssh-sha2";
   description
      "Elliptic Curve Diffie-Hellman (ECDH) key exchange using the
      nistp521 curve and the SHA2 family of hashing algorithms.";
reference
   "RFC 5656: Elliptic Curve Algorithm Integration in the
   Secure Shell Transport Layer";
}

identity encryption-alg-base {
   description
      "Base identity used to identify encryption algorithms.";
}

identity triple-des-cbc {
   base encryption-alg-base;
   description
      "Three-key 3DES in CBC mode.";
reference
   "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
}

identity aes128-cbc {
   base encryption-alg-base;
   description
      "AES in CBC mode, with a 128-bit key.";
reference
   "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
}

identity aes192-cbc {
   base encryption-alg-base;
   description
      "AES in CBC mode, with a 192-bit key.";
reference
   "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
}
identity aes256-cbc {
    base encryption-alg-base;
    description
        "AES in CBC mode, with a 256-bit key.";
    reference
        "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
}

identity aes128-ctr {
    base encryption-alg-base;
    if-feature ssh-ctr;
    description
        "AES in SDCTR mode, with 128-bit key.";
    reference
        "RFC 4344: The Secure Shell (SSH) Transport Layer Encryption Modes";
}

identity aes192-ctr {
    base encryption-alg-base;
    if-feature ssh-ctr;
    description
        "AES in SDCTR mode, with 192-bit key.";
    reference
        "RFC 4344: The Secure Shell (SSH) Transport Layer Encryption Modes";
}

identity aes256-ctr {
    base encryption-alg-base;
    if-feature ssh-ctr;
    description
        "AES in SDCTR mode, with 256-bit key.";
    reference
        "RFC 4344: The Secure Shell (SSH) Transport Layer Encryption Modes";
}

identity mac-alg-base {
    description
        "Base identity used to identify message authentication code (MAC) algorithms.";
}

identity hmac-shal {
    base mac-alg-base;
    description
        "HMAC-SHA1";
}
reference
  "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
}

identity hmac-sha2-256 {
  base mac-alg-base;
  if-feature "ssh-sha2";
  description
    "HMAC-SHA2-256";
  reference
    "RFC 6668: SHA-2 Data Integrity Verification for the
    Secure Shell (SSH) Transport Layer Protocol";
}

identity hmac-sha2-512 {
  base mac-alg-base;
  if-feature "ssh-sha2";
  description
    "HMAC-SHA2-512";
  reference
    "RFC 6668: SHA-2 Data Integrity Verification for the
    Secure Shell (SSH) Transport Layer Protocol";
}

// groupings

grouping transport-params-grouping {
  description
    "A reusable grouping for SSH transport parameters.";
  reference
    "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
  container host-key {
    description
      "Parameters regarding host key.";
    leaf-list host-key-alg {
      type identityref {
        base public-key-alg-base;
      }
      ordered-by user;
      description
        "Acceptable host key algorithms in order of descending preference.

        If this leaf-list is not configured (has zero elements) the
        acceptable host key algorithms are implementation-defined.";
    }
  }
}
container key-exchange {
  description
    "Parameters regarding key exchange.";
  leaf-list key-exchange-alg {
    type identityref {
      base key-exchange-alg-base;
    }
    ordered-by user;
    description
      "Acceptable key exchange algorithms in order of descending preference.

      If this leaf-list is not configured (has zero elements)
      the acceptable key exchange algorithms are implementation defined.";
  }
}

container encryption {
  description
    "Parameters regarding encryption.";
  leaf-list encryption-alg {
    type identityref {
      base encryption-alg-base;
    }
    ordered-by user;
    description
      "Acceptable encryption algorithms in order of descending preference.

      If this leaf-list is not configured (has zero elements)
      the acceptable encryption algorithms are implementation defined.";
  }
}

container mac {
  description
    "Parameters regarding message authentication code (MAC).";
  leaf-list mac-alg {
    type identityref {
      base mac-alg-base;
    }
    ordered-by user;
    description
      "Acceptable MAC algorithms in order of descending preference.

      If this leaf-list is not configured (has zero elements)
      the acceptable MAC algorithms are implementation-
6. Security Considerations

The YANG modules defined in this document are designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

Since the modules defined in this document define only groupings, these considerations are primarily for the designers of other modules that use these groupings.

There are a number of data nodes defined in the YANG modules that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

/: The entire data tree defined by all the modules defined in this draft are sensitive to write operations. For instance, the addition or removal of references to keys, certificates, trusted anchors, etc., can dramatically alter the implemented security policy. However, no NACM annotations are applied as the data SHOULD be editable by users other than a designated ‘recovery session’.

Some of the readable data nodes in the YANG modules may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:
/client-auth/password: This node in the ‘ietf-ssh-client’ module is additionally sensitive to read operations such that, in normal use cases, it should never be returned to a client. The only time this node should be returned is to support backup/restore type workflows. However, no NACM annotations are applied as the data SHOULD be writable by users other than a designated ‘recovery session’.

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

NONE

7. IANA Considerations

7.1. The IETF XML Registry

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

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

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

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

7.2. The YANG Module Names Registry

This document registers three YANG modules in the YANG Module Names registry [RFC7950]. Following the format in [RFC7950], the following registrations are requested:
name:         ietf-ssh-client
prefix:       sshc
reference:    RFC XXXX

name:         ietf-ssh-server
prefix:       sshs
reference:    RFC XXXX

name:         ietf-ssh-common
prefix:       sshcmn
reference:    RFC XXXX

8. Acknowledgements

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

9. References

9.1. Normative References

[I-D.ietf-netconf-keystore]
Watsen, K., "YANG Data Model for a "Keystore" Mechanism",
draft-ietf-netconf-keystore-04 (work in progress), October
2017.

[I-D.ietf-netconf-trust-anchors]
Watsen, K., "YANG Data Model for Global Trust Anchors",
draft-ietf-netconf-trust-anchors-00 (work in progress),
June 2018.

[RFC2119]  Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,

[RFC4344]  Bellare, M., Kohno, T., and C. Namprempre, "The Secure
Shell (SSH) Transport Layer Encryption Modes", RFC 4344,
DOI 10.17487/RFC4344, January 2006,


9.2. Informative References


Appendix A. Change Log

A.1. 00 to 01

- Noted that '0.0.0.0' and '::' might have special meanings.
- Renamed "keychain" to "keystore".

A.2. 01 to 02

- Removed the groupings 'listening-ssh-client-grouping' and 'listening-ssh-server-grouping'. Now modules only contain the transport-independent groupings.
- Simplified the "client-auth" part in the ietf-ssh-client module. It now inlines what it used to point to keystore for.
- Added cipher suites for various algorithms into new 'ietf-ssh-common' module.

A.3. 02 to 03

- Removed 'RESTRICTED' enum from 'password' leaf type.
- Added a 'must' statement to container 'server-auth' asserting that at least one of the various auth mechanisms must be specified.
- Fixed description statement for leaf 'trusted-ca-certs'.

A.4. 03 to 04

- Change title to "YANG Groupings for SSH Clients and SSH Servers"
- Added reference to RFC 6668
- Added RFC 8174 to Requirements Language Section.
- Enhanced description statement for ietf-ssh-server's "trusted-ca-certs" leaf.
- Added mandatory true to ietf-ssh-client’s "client-auth" 'choice' statement.
- Changed the YANG prefix for module ietf-ssh-common from 'sshcom' to 'sshcmn'.
- Removed the compression algorithms as they are not commonly configurable in vendors’ implementations.
o Updating descriptions in transport-params-grouping and the servers’s usage of it.

o Now tree diagrams reference ietf-netmod-yang-tree-diagrams

o Updated YANG to use typedefs around leafrefs to common keystore paths

o Now inlines key and certificates (no longer a leafref to keystore)

A.5. 04 to 05

o Merged changes from co-author.

A.6. 05 to 06

o Updated to use trust anchors from trust-anchors draft (was keystore draft)

o Now uses new keystore grouping enabling asymmetric key to be either locally defined or a reference to the keystore.

Authors’ Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Gary Wu
Cisco Systems

EMail: garywu@cisco.com
Abstract

This document defines a YANG data model and associated mechanisms enabling subscriber-specific subscriptions to a publisher’s event streams. Applying these elements allows a subscriber to request for and receive a continuous, custom feed of publisher generated information.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at 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 February 4, 2019.

Copyright Notice

Copyright (c) 2018 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
1. Introduction

This document defines a YANG data model and associated mechanisms enabling subscriber-specific subscriptions to a publisher’s event streams. Effectively this enables a ‘subscribe then publish’ capability where the customized information needs and access
permissions of each target receiver are understood by the publisher before subscribed event records are marshaled and pushed. The receiver then gets a continuous, custom feed of publisher generated information.

While the functionality defined in this document is transport-agnostic, transports like NETCONF [RFC6241] or RESTCONF [RFC8040] can be used to configure or dynamically signal subscriptions, and there are bindings defined for subscribed event record delivery for NETCONF within [I-D.draft-ietf-netconf-netconf-event-notifications], and for HTTP2 or HTTP1.1 within [I-D.draft-ietf-netconf-restconf-notif].

The YANG model in this document conforms to the Network Management Datastore Architecture defined in [RFC8342].

1.1. Motivation

Various limitations in [RFC5277] are discussed in [RFC7923]. Resolving these issues is the primary motivation for this work. Key capabilities supported by this document include:

- multiple subscriptions on a single transport session
- support for dynamic and configured subscriptions
- modification of an existing subscription in progress
- per-subscription operational counters
- negotiation of subscription parameters (through the use of hints returned as part of declined subscription requests)
- subscription state change notifications (e.g., publisher driven suspension, parameter modification)
- independence from transport

1.2. 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Client: defined in [RFC8342].

Configuration: defined in [RFC8342].
Configuration datastore: defined in [RFC8342].

Configured subscription: A subscription installed via configuration into a configuration datastore.

Dynamic subscription: A subscription created dynamically by a subscriber via a remote procedure call.

Event: An occurrence of something that may be of interest. Examples include a configuration change, a fault, a change in status, crossing a threshold, or an external input to the system.

Event occurrence time: a timestamp matching the time an originating process identified as when an event happened.

Event record: A set of information detailing an event.

Event stream: A continuous, chronologically ordered set of events aggregated under some context.

Event stream filter: Evaluation criteria which may be applied against event records within an event stream. Event records pass the filter when specified criteria are met.

Notification message: Information intended for a receiver indicating that one or more event(s) have occurred.

Publisher: An entity responsible for streaming notification messages per the terms of a subscription.

Receiver: A target to which a publisher pushes subscribed event records. For dynamic subscriptions, the receiver and subscriber are the same entity.

Subscriber: A client able to request and negotiate a contract for the generation and push of event records from a publisher. For dynamic subscriptions, the receiver and subscriber are the same entity.

Subscription: A contract with a publisher, stipulating which information one or more receivers wish to have pushed from the publisher without the need for further solicitation.

All YANG tree diagrams used in this document follow the notation defined in [RFC8340].
1.3. Solution Overview

This document describes a transport agnostic mechanism for subscribing to and receiving content from an event stream within a publisher. This mechanism is through the use of a subscription.

Two types of subscriptions are supported:

1. Dynamic subscriptions, where a subscriber initiates a subscription negotiation with a publisher via an RPC. If the publisher is able to serve this request, it accepts it, and then starts pushing notification messages back to the subscriber. If the publisher is not able to serve it as requested, then an error response is returned. This response MAY include hints at subscription parameters that, had they been present, would have enabled the dynamic subscription request to be accepted.

2. Configured subscriptions, which allow the management of subscriptions via a configuration so that a publisher can send notification messages to a receiver of a configured subscription. Support for configured subscriptions is optional, with its availability advertised via a YANG feature.

Additional characteristics differentiating configured from dynamic subscriptions include:

- The lifetime of a dynamic subscription is bound by the transport session used to establish it. For connection-oriented stateful transports like NETCONF, the loss of the transport session will result in the immediate termination of any associated dynamic subscriptions. For connectionless or stateless transports like HTTP, a lack of receipt acknowledgment of a sequential set of notification messages and/or keep-alives can be used to trigger a termination of a dynamic subscription. Contrast this to the lifetime of a configured subscription. This lifetime is driven by relevant configuration being present within the publisher’s applied configuration. Being tied to configuration operations implies configured subscriptions can be configured to persist across reboots, and implies a configured subscription can persist even when its publisher is fully disconnected from any network.

- Configured subscriptions can be modified by any configuration client with write permission on the configuration of the subscription. Dynamic subscriptions can only be modified via an RPC request made by the original subscriber, or a change to configuration data referenced by the subscription.
Note that there is no mixing-and-matching of dynamic and configured operations on a single subscription. Specifically, a configured subscription cannot be modified or deleted using RPCs defined in this document. Similarly, a subscription established via RPC cannot be modified through configuration operations. Also note that transport specific transport drafts based on this specification MUST detail the life cycles of both dynamic and configured subscriptions.

A publisher MAY terminate a dynamic subscription at any time. Similarly, it MAY decide to temporarily suspend the sending of notification messages for any dynamic subscription, or for one or more receivers of a configured subscription. Such termination or suspension is driven by internal considerations of the publisher.

1.4. Relationship to RFC-5277

This document is intended to provide a superset of the subscription capabilities initially defined within [RFC5277]. Especially when extending an existing [RFC5277] implementation, it is important to understand what has been reused and what has been replaced. Key relationships between these two documents include:

- This document defines a transport independent capability, [RFC5277] is specific to NETCONF.
- The data model in this document is used instead of the data model in Section 3.4 of [RFC5277] for the new operations.
- The RPC operations in this draft replaces the operation "create-subscription" defined in [RFC5277], section 4.
- The <notification> message of [RFC5277], Section 4 is used.
- The included contents of the "NETCONF" event stream are identical between this document and [RFC5277].
- A publisher MAY implement both the Notification Management Schema and RPCs defined in [RFC5277] and this new document concurrently.
- Unlike [RFC5277], this document enables a single transport session to intermix of notification messages and RPCs for different subscriptions.

2. Solution

Per the overview provided in Section 1.3, this section details the overall context, state machines, and subsystems which may be assembled to allow the subscription of events from a publisher.
2.1. Event Streams

An event stream is a named entity on a publisher which exposes a continuously updating set of event records. Each event stream is available for subscription. It is out of the scope of this document to identify a) how streams are defined (other than the NETCONF stream), b) how event records are defined/generated, and c) how event records are assigned to streams.

There is only one reserved event stream name within this document: "NETCONF". The "NETCONF" event stream contains all NETCONF XML event record information supported by the publisher, except for the subscription state notifications described in Section 2.7. Among these included NETCONF XML event records are individual YANG 1.1 notifications described in section 7.16 of [RFC7950]. Each of these YANG 1.1 notifications will be treated as a distinct event record. Beyond the "NETCONF" stream, implementations MAY define additional event streams.

As event records are created by a system, they may be assigned to one or more streams. The event record is distributed to a subscription’s receiver(s) where: (1) a subscription includes the identified stream, and (2) subscription filtering does not exclude the event record from that receiver.

Access control permissions may be used to silently exclude event records from within an event stream for which the receiver has no read access. As an example of how this might be accomplished, see [RFC8341] section 3.4.6. Note that per Section 2.7 of this document, subscription state change notifications are never filtered out.

If no access control permissions are in place for event records on an event stream, then a receiver MUST be allowed access to all the event records. If subscriber permissions change during the lifecycle of a subscription and event stream access is no longer permitted, then the subscription MUST be terminated.

Event records MUST NOT be delivered to a receiver in a different order than they were placed onto an event stream.

2.2. Event Stream Filters

This document defines an extensible filtering mechanism. The filter itself is a boolean test which is placed on the content of an event record. A 'false' filtering result causes the event message to be excluded from delivery to a receiver. A filter never results in information being stripped from within an event record prior to that event record being encapsulated within a notification message. The
two optional event stream filtering syntaxes supported are [XPATH]
and subtree [RFC6241].

If no event stream filter is provided within a subscription, all
event records on an event stream are to be sent.

2.3. QoS

This document provide for several QoS parameters. These parameters
indicate the treatment of a subscription relative to other traffic
between publisher and receiver. Included are:

- A "dscp" marking to differentiate prioritization of notification
  messages during network transit.
- A "weighting" so that bandwidth proportional to this weighting can
  be allocated to this subscription relative to other subscriptions
  destined for that receiver.
- A "dependency" upon another subscription.

If the publisher supports the "dscp" feature, then a subscription
with a "dscp" leaf MUST result in a corresponding [RFC2474] DSCP
marking being placed within the IP header of any resulting
notification messages and subscription state change notifications.

For the "weighting" parameter, when concurrently dequeuing
notification messages from multiple subscriptions to a receiver, the
publisher MUST allocate bandwidth to each subscription proportionally
to the weights assigned to those subscriptions. "Weighting" is an
optional capability of the publisher; support for it is identified
via the "qos" feature.

If a subscription has the "dependency" parameter set, then any
buffered notification messages containing event records selected by
the parent subscription MUST be dequeued prior to the notification
messages of the dependent subscription. If notification messages
have dependencies on each other, the notification message queued the
longest MUST go first. If a "dependency" included within an RPC
references a subscription which does not exist or is no longer
accessible to that subscriber, that "dependency" MUST be silently
removed. "Dependency" is an optional capability of the publisher;
support for it is identified via the "qos" feature.
2.4. Dynamic Subscriptions

Dynamic subscriptions are managed via protocol operations (in the form of [RFC7950], Section 7.14 RPCs) made against targets located within the publisher. These RPCs have been designed extensibly so that they may be augmented for subscription targets beyond event streams. For examples of such augmentations, see the RPC augmentations within [I-D.ietf-netconf-yang-push]'s YANG model.

2.4.1. Dynamic Subscription State Model

Below is the publisher’s state machine for a dynamic subscription. Each state is shown in its own box. It is important to note that such a subscription doesn’t exist at the publisher until an "establish-subscription" RPC is accepted. The mere request by a subscriber to establish a subscription is insufficient for that subscription to be externally visible. Start and end states are depicted to reflect subscription creation and deletion events.

```
............
|           : start :
|           :.......:
| establish-subscription
|                   |-------modify-subscription--------.
|                   | .--receiver active --insufficient CPU, b/w-->| receiver suspended |
|                   |<-----CPU, b/w sufficient--|............
| receiver          | delete/kill-subscription
| suspend/kill-subscription

Figure 1: Publisher’s state for a dynamic subscription
```

Of interest in this state machine are the following:

- Successful "establish-subscription" or "modify-subscription" RPCs put the subscription into the active state.
2.4.2. Establishing a Dynamic Subscription

The "establish-subscription" RPC allows a subscriber to request the creation of a subscription. The transport selected by the subscriber to reach the publisher MUST be able to support multiple "establish-subscription" requests made within the same transport session.

The input parameters of the operation are:

- A "stream" name which identifies the targeted event stream against which the subscription is applied.
- An event stream filter which may reduce the set of event records pushed.
- Where the transport used by the RPC supports multiple encodings, an optional "encoding" for the event records pushed. Note: If no "encoding" is included, the encoding of the RPC MUST be used.
- An optional "stop-time" for the subscription. If no "stop-time" is present, notification messages will continue to be sent until the subscription is terminated.
- An optional "start-time" for the subscription. The "start-time" MUST be in the past and indicates that the subscription is requesting a replay of previously generated information from the...
event stream. For more on replay, see Section 2.4.2.1. Where there is no "start-time", the subscription starts immediately.

If the publisher can satisfy the "establish-subscription" request, it replies with an identifier for the subscription, and then immediately starts streaming notification messages.

Below is a tree diagram for "establish-subscription". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---x establish-subscription
    +---w input
        |   +---w (target)
        |       +---(stream)
        |           +---w (stream-filter)?
        |           |   +---w stream-filter-ref
        |           |     stream-filter-ref
        |           |     +---w (within-subscription)
        |           |     +---w (filter-spec)?
        |           |     |   +---w stream-subtree-filter? <anydata>
        |           |     |       {subtree}?
        |           |     +---w stream-xpath-filter?
        |           |     |     yang:xpath1.0 {xpath}?
        |           |     +---w stream-ref
        |           |     +---w replay-start-time? yang:date-and-time {replay}?
        |           +---w stop-time? yang:date-and-time
        |           +---w dscp? inet:dscp {dscp}?
        +---w weighting? uint8 {qos}?
    +---w replay-start-time? yang:date-and-time {replay}?
    +---w dependency? subscription-id {qos}?
    +---w encoding? encoding

output
+---ro identifier subscription-id
+---ro replay-start-time-revision? yang:date-and-time {replay}?
```

Figure 2: establish-subscription RPC tree diagram

A publisher MAY reject the "establish-subscription" RPC for many reasons as described in Section 2.4.6. The contents of the resulting RPC error response MAY include details on input parameters which if considered in a subsequent "establish-subscription" RPC, may result in a successful subscription establishment. Any such hints MUST be
transported within a yang-data "establish-subscription-stream-error-info" container included within the RPC error response.

```
yang-data establish-subscription-stream-error-info
   +++-ro reason?                   identityref
   +++-ro filter-failure-hint?      string
```

Figure 3: establish-subscription RPC yang-data tree diagram

2.4.2.1. Requesting a replay of event records

Replay provides the ability to establish a subscription which is also capable of passing recently generated event records. In other words, as the subscription initializes itself, it sends any previously generated content from within the target event stream which meets the filter and timeframe criteria. The end of these historical event records is identified via a "replay-completed" state change notification. Any event records generated since the subscription establishment may then follow. For a particular subscription, all event records will be delivered in the order they are placed into the stream.

Replay is an optional feature which is dependent on an event stream supporting some form of logging. This document puts no restrictions on the size or form of the log, where it resides within the publisher, or when event record entries in the log are purged.

The inclusion of a "replay-start-time" within an "establish-subscription" RPC indicates a replay request. If the "replay-start-time" contains a value that is earlier than what a publisher’s retained history supports, then if the subscription is accepted, the actual publisher’s revised start time MUST be set in the returned "replay-start-time-revision" object.

A "stop-time" parameter may be included in a replay subscription. For a replay subscription, the "stop-time" MAY be earlier than the current time, but MUST be later than the "replay-start-time".

If the time the replay starts is later than the time marked within any event records retained within the replay buffer, then the publisher MUST send a "replay-completed" notification immediately after a successful establish-subscription RPC response.

If an event stream supports replay, the "replay-support" leaf is present in the "/streams/stream" list entry for the stream. An event stream that does support replay is not expected to have an unlimited supply of saved notifications available to accommodate any given
replay request. To assess the timeframe available for replay, subscribers can read the leafs "replay-log-creation-time" and "replay-log-aged-time". See Figure 18 for the YANG tree, and Section 4 for the YANG model describing these elements. The actual size of the replay log at any given time is a publisher specific matter. Control parameters for the replay log are outside the scope of this document.

2.4.3. Modifying a Dynamic Subscription

The "modify-subscription" operation permits changing the terms of an existing dynamic subscription. Dynamic subscriptions can be modified any number of times. If the publisher accepts the requested modifications, it acknowledges success to the subscriber, then immediately starts sending event records based on the new terms.

Subscriptions created by configuration cannot be modified via this RPC. However configuration may be used to modify objects referenced by the subscription (such as a referenced filter).

Below is a tree diagram for "modify-subscription". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---x modify-subscription
   +---w input
   |    +---w identifier               subscription-id
   |    +---w (target)
   |         +---w (stream)
   |             +---w (stream-filter)?
   |                 +---:(by-reference)
   |                 |     +---w stream-filter-ref
   |                 |       stream-filter-ref
   |                 +---:(within-subscription)
   |                     +---w (filter-spec)?
   |                     |    +---w stream-subtree-filter? <anydata>
   |                     |          {subtree}? (subtree)?
   |                     +---:(stream-xpath-filter)
   |                             +---w stream-xpath-filter? yang:xpath1.0 {xpath}?
   |                             yang:date-and-time
   +---w stop-time?
```

Figure 4: modify-subscription RPC tree diagram

If the publisher accepts the requested modifications on a currently suspended subscription, the subscription will immediately be resumed (i.e., the modified subscription is returned to the active state.)
The publisher MAY immediately suspend this newly modified subscription through the "subscription-suspended" notification before any event records are sent.

If the publisher rejects the RPC request, the subscription remains as prior to the request. That is, the request has no impact whatsoever. Rejection of the RPC for any reason is indicated by via RPC error as described in Section 2.4.6. The contents of such a rejected RPC MAY include hints on inputs which (if considered) may result in a successfully modified subscription. These hints MUST be transported within a yang-data "modify-subscription-stream-error-info" container inserted into the RPC error response.

Below is a tree diagram for "modify-subscription-RPC-yang-data". All objects contained in this tree are described within the included YANG model within Section 4.

```
yang-data modify-subscription-stream-error-info
   +--ro modify-subscription-stream-error-info
   +--ro reason? identityref
   +--ro filter-failure-hint? string
```

Figure 5: modify-subscription RPC yang-data tree diagram

2.4.4. Deleting a Dynamic Subscription

The "delete-subscription" operation permits canceling an existing subscription. If the publisher accepts the request, and the publisher has indicated success, the publisher MUST NOT send any more notification messages for this subscription. If the delete request matches a known subscription established on the same transport session, then it MUST be deleted; otherwise it MUST be rejected with no changes to the publisher.

Below is a tree diagram for "delete-subscription". All objects contained in this tree are described within the included YANG model within Section 4.

```
---x delete-subscription
   +--w input
     +--w identifier subscription-id
```

Figure 6: delete-subscription RPC tree diagram

Dynamic subscriptions can only be deleted via this RPC using the same transport session previously used for subscription establishment. Configured subscriptions cannot be deleted using RPCs.
2.4.5. Killing a Dynamic Subscription

The "kill-subscription" operation permits an operator to end a dynamic subscription which is not associated with the transport session used for the RPC. A publisher MUST terminate any dynamic subscription identified by RPC request.

Configured subscriptions cannot be killed using this RPC. Instead, configured subscriptions are deleted as part of regular configuration operations. Publishers MUST reject any RPC attempt to kill a configured subscription.

Below is a tree diagram for "kill-subscription". All objects contained in this tree are described within the included YANG model within Section 4.

```
  +---x kill-subscription
        +---w input
             +---w identifier subscription-id
```

Figure 7: kill-subscription RPC tree diagram

2.4.6. RPC Failures

Whenever an RPC is unsuccessful, the publisher returns relevant information as part of the RPC error response. Transport level error processing MUST be done before RPC error processing described in this section. In all cases, RPC error information returned will use existing transport layer RPC structures, such as those seen with NETCONF in [RFC6241] Appendix A, or with RESTCONF in [RFC8040] Section 7.1. These structures MUST be able to encode subscription specific errors identified below and defined within this document’s YANG model.

As a result of this mixture, how subscription errors are encoded within an RPC error response is transport dependent. Following are valid errors which can occur for each RPC:
To see a NETCONF based example of an error response from above, see [I-D.draft-ietf-netconf-netconf-event-notifications], Figure 10.

There is one final set of transport independent RPC error elements included in the YANG model. These are the following three yang-data structures for failed event stream subscriptions:

1. "establish-subscription-stream-error-info": This MUST be returned if an RPC error reason has not been placed elsewhere within the transport portion of a failed "establish-subscription" RPC response. This MUST be sent if hints on how to overcome the RPC error are included.

2. "modify-subscription-stream-error-info": This MUST be returned if an RPC error reason has not been placed elsewhere within the transport portion of a failed "modify-subscription" RPC response. This MUST be sent if hints on how to overcome the RPC error are included.

3. "delete-subscription-error-info": This MUST be returned if an RPC error reason has not been placed elsewhere within the transport portion of a failed "delete-subscription" or "kill-subscription" RPC response.

2.5. Configured Subscriptions

A configured subscription is a subscription installed via configuration. Configured subscriptions may be modified by any configuration client with the proper permissions. Subscriptions can be modified or terminated via configuration at any point of their lifetime. Multiple configured subscriptions MUST be supportable over a single transport session.

Configured subscriptions have several characteristics distinguishing them from dynamic subscriptions:
o persistence across publisher reboots,

o persistence even when transport is unavailable, and

o an ability to send notification messages to more than one receiver (note that receivers are unaware of the existence of any other receivers.)

On the publisher, supporting configured subscriptions is optional and advertised using the "configured" feature. On a receiver of a configured subscription, support for dynamic subscriptions is optional except where replaying missed event records is required.

In addition to the subscription parameters available to dynamic subscriptions described in Section 2.4.2, the following additional parameters are also available to configured subscriptions:

o A "transport" which identifies the transport protocol to use to connect with all subscription receivers.

o One or more receivers, each intended as the destination for event records. Note that each individual receiver is identifiable by its "name". Via this "name", publisher transport parameters must be defined in order to establish and maintain a transport connection with a receiver. This transport specific reference can come in several forms, including the augmentation of leafrefs to an actual transport instance. Such augmentations must be defined in transport specific specifications building upon this document.

o Optional parameters to identify where traffic should egress a publisher:

* A "source-interface" which identifies the egress interface to use from the publisher. Publisher support for this is optional and advertised using the "interface-designation" feature.

* A "source-address" address, which identifies the IP address to stamp on notification messages destined for the receiver.

* A "source-vrf" which identifies the VRF on which to reach receivers. This VRF is a network instance as defined within [I-D.draft-ietf-rtgwg-ni-model]. Publisher support for VRFs is optional and advertised using the "supports-vrf" feature.

If none of the above parameters are set, notification messages MUST egress the publisher’s default interface.
A tree diagram describing these parameters is shown in Figure 20 within Section 3.3. All parameters are described within the YANG model in Section 4.

2.5.1. Configured Subscription State Model

Below is the state machine for a configured subscription on the publisher. This state machine describes the three states (valid, invalid, and concluded), as well as the transitions between these states. Start and end states are depicted to reflect configured subscription creation and deletion events. The creation or modification of a configured subscription initiates an evaluation by the publisher to determine if the subscription is in valid or invalid states. The publisher uses its own criteria in making this determination. If in the valid state, the subscription becomes operational. See (1) in the diagram below.

```
............
: start :-.
:........: |
    create |--modify----|--------|--------|--------|
       V  V   |--------|       |--------|       |--------|
[-evaluate]--no--->invalid|delete|: end ;<delete--concluded|
       ^                      |.......|         |.......|
[-evaluate]--no-(2)       ^                      |.......|
     yes       ^
    modify     ^
      (subscription-     ^
        terminated*)     ^
      (subscription-     ^
        terminated*)     ^
      (subscription-     ^
        concluded*)
(1)       (3)         (4)        (5)
'-->|                          |
     |--------------------------|
     | valid                     |
     |--------------------------|
```

Legend:
dotted boxes: subscription added or removed via configuration
dashed boxes: states for a subscription
[evaluate]: decision point on whether the subscription is supportable
(*) : resulting subscription state change notification

Figure 8: Publisher state model for a configured subscription

A subscription in the valid state may move to the invalid state in one of two ways. First, it may be modified in a way which fails a re-evaluation. See (2) in the diagram. Second, the publisher might determine that the subscription is no longer supportable. This could
be for reasons of an unexpected but sustained increase in an event stream's event records, degraded CPU capacity, a more complex referenced filter, or other higher priority subscriptions which have usurped resources. See (3) in the diagram. No matter the case, a "subscription-terminated" notification is sent to any receivers in an active or suspended state. A subscription in the valid state may also transition to the concluded state via (5) if a configured stop time has been reached. In this case, a "subscription-concluded" notification is sent to any receivers in active or suspended states. Finally, a subscription may be deleted by configuration (4).

When a subscription is in the valid state, a publisher will attempt to connect with all receivers of a configured subscription and deliver notification messages. Below is the state machine for each receiver of a configured subscription. This receiver state machine is fully contained within the state machine of the configured subscription, and is only relevant when the configured subscription is in the valid state.

```
<table>
<thead>
<tr>
<th>valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>receiver</td>
</tr>
<tr>
<td>connecting</td>
</tr>
<tr>
<td>&lt;-timeout----------------&gt; receiver</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>'-----&gt;-----</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>'-----'</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>'--------'</td>
</tr>
</tbody>
</table>
```

Legend:
dashed boxes which include the word 'receiver' show the possible states for an individual receiver of a valid configured subscription. * indicates a state change notification

Figure 9: Receiver state for a configured subscription on a Publisher

When a configured subscription first moves to the valid state, the "state" leaf of each receiver is initialized to "connecting". If transport connectivity is not available to any receiver and there are any notification messages to deliver, a transport session is established (e.g., through [RFC8071]). Individual receivers are
moved to the active state when a "subscription-started" state change
notification is successfully passed to that receiver (a). Event
records are only sent to active receivers. Receivers of a configured
subscription remain active if both transport connectivity can be
verified to the receiver, and event records are not being dropped due
to a publisher buffer overflow. The result is that a receiver will
remain active on the publisher as long as events aren’t being lost,
or the receiver cannot be reached. In addition, a configured
subscription’s receiver MUST be moved to connecting if transport
connectivity cannot be achieved, or if the receiver is reset via the
"reset" action (b), (c). For more on reset, see Section 2.5.5.

A configured subscription’s receiver MUST be moved to the suspended
state if there is transport connectivity between the publisher and
receiver, but notification messages are failing to be delivered due
to publisher buffer overflow, or notification messages are not able
to be generated for that receiver due to insufficient CPU (d). This
is indicated to the receiver by the "subscription-suspended" state
change notification.

A configured subscription receiver MUST be returned to the active
state from the suspended state when notification messages are able to
be generated, bandwidth is sufficient to handle the notification
messages, and a receiver has successfully been sent a "subscription-
resumed" or "subscription-modified" state change notification (e).
The choice as to which of these two state change notifications is
sent is determined by whether the subscription was modified during
the period of suspension.

Modification of a configured subscription is possible at any time. A
"subscription-modified" state change notification will be sent to all
active receivers, immediately followed by notification messages
conforming to the new parameters. Suspended receivers will also be
informed of the modification. However this notification will await
the end of the suspension for that receiver (e).

The mechanisms described above are mirrored in the RPCs and
notifications within the document. It should be noted that these
RPCs and notifications have been designed to be extensible and allow
subscriptions into targets other than event streams. For instance,
the YANG module defined in Section 5 of [I-D.ietf-netconf-yang-push]
augments "/sn:modify-subscription/sn:input/sn:target".

2.5.2. Creating a Configured Subscription

Configured subscriptions are established using configuration
operations against the top-level "subscriptions" subtree.
Because there is no explicit association with an existing transport session, configuration operations MUST include additional parameters beyond those of dynamic subscriptions to indicate each receiver, how to contact that receiver, and possibly whether the notification messages need to come from a specific egress interface on the publisher. Some of these parameters MUST be configured via transport specific augmentations to this specification.

After a subscription is successfully established, the publisher immediately sends a "subscription-started" state change notification to each receiver. It is quite possible that upon configuration, reboot, or even steady-state operations, a transport session may not be currently available to the receiver. In this case, when there is something to transport for an active subscription, transport specific call-home operations will be used to establish the connection. When transport connectivity is available, notification messages may then be pushed.

With active configured subscriptions, it is allowable to buffer event records even after a "subscription-started" has been sent. However if events are lost (rather than just delayed) due to replay buffer overflow, a new "subscription-started" must be sent. This new "subscription-started" indicates an event record discontinuity.

To see an example of subscription creation using configuration operations over NETCONF, see Appendix A of [I-D.draft-ietf-netconf-netconf-event-notifications].

Note that it is possible to configure replay on a configured subscription. This capability is to allow a configured subscription to exist on a system so that event records generated during and following boot can be buffered and pushed as soon as the transport session is established.

2.5.3. Modifying a Configured Subscription

Configured subscriptions can be modified using configuration operations against the top-level "subscriptions" subtree.

If the modification involves adding receivers, added receivers are placed in the connecting state. If a receiver is removed, the state change notification "subscription-terminated" is sent to that receiver if that receiver is active or suspended.

If the modification involves changing the policies for the subscription, the publisher sends to currently active receivers a "subscription-modified" notification. For any suspended receivers, a "subscription-modified" notification will be delayed until the
receiver is resumed. (Note: in this case, the "subscription-modified" notification informs the receiver that the subscription has been resumed, so no additional "subscription-resumed" need be sent. Also note that if multiple modifications have occurred during the suspension, only the latest one need be sent to the receiver.)

2.5.4. Deleting a Configured Subscription

Subscriptions can be deleted through configuration against the top-level "subscriptions" subtree.

Immediately after a subscription is successfully deleted, the publisher sends to all receivers of that subscription a state change notification stating the subscription has ended (i.e., "subscription-terminated").

2.5.5. Resetting a Configured Subscription Receiver

It is possible that a configured subscription to a receiver needs to be reset. This is accomplished via the "reset" action within the YANG model at "/subscriptions/subscription/receivers/receiver/reset". This re-initialization may be useful in cases where a publisher has timed out trying to reach a receiver. When such a reset occurs, a transport session will be initiated if necessary, and a new "subscription-started" notification will be sent. This action does not have any effect on transport connectivity if the needed connectivity already exists.

2.5.6. Replay for a Configured Subscription

It is possible to do replay on a configured subscription. This is supported via the configuration of the "configured-replay" object on the subscription. The setting of this object enables the streaming of the buffered events for the subscribed stream. All buffered event which have been retained since the last publisher restart will be sent.

Replay of events records created since restart is useful. It allows event records generated before transport connectivity establishment to be passed to a receiver. Setting the restart time as the earliest configured replay time precludes possibility of resending of event records logged prior to publisher restart. It also ensures the same records will be sent to each configured receiver, regardless of the speed of transport connectivity establishment to each receiver. Finally, establishing restart as the earliest potential time for event records to be included within notification messages, a well-understood timeframe for replay is defined.
As a result, when any configured subscription receivers become active, buffered event records will be sent immediately after the "subscription-started" notification. The leading event record sent will be the first event record subsequent to the latest of three different times: the "replay-log-creation-time", "replay-log-aged-time", or the most recent publisher boot time. The "replay-log-creation-time" and "replay-log-aged-time" are discussed in Section 2.4.2.1. The most recent publisher boot time ensures that duplicate event records are not replayed from a previous time the publisher was booted.

It is quite possible that a receiver might want to retrieve event records from a stream prior to the latest boot. If such records exist where there is a configured replay, the publisher MUST send the time of the event record immediately preceding the "replay-start-time" within the "replay-previous-event-time" leaf. Through the existence of the "replay-previous-event-time", the receiver will know that earlier events prior to reboot exist. In addition, if the subscriber was previously receiving event records with the same subscription id, the receiver can determine if there was a timegap where records generated on the publisher were not successfully received. And with this information, the receiver may choose to dynamically subscribe to retrieve any event records placed into the stream before the most recent boot time.

All other replay functionality remains the same as with dynamic subscriptions as described in Section 2.4.2.1.

2.5.7. Transport Connectivity for a Configured Subscription

This specification is transport independent. However supporting a configured subscription will often require the establishment of transport connectivity. And the parameters used for this transport connectivity establishment happen to be transport specific. As a result, the YANG model defined within Section 4 is not able to directly define these transport parameters.

It is necessary to support the connection establishment process. To support this function, the YANG model does include a node where transport specific parameters may be augmented into the model. This node is the choice "transport" located under "/subscriptions/subscription/receivers/receiver". By augmenting YANG case entries under this node, system developers are able to incorporate the YANG objects necessary to support the transport connectivity establishment process.

Any implementation of this specification where the optional feature "configured" is supported MUST augment the YANG model with at least
one transport specific YANG case under the choice "transport". For more information and an example on how this might be accomplished see Appendix A.

2.6. Event Record Delivery

Whether dynamic or configured, once a subscription has been set up, the publisher streams event records via notification messages per the terms of the subscription. For dynamic subscriptions, notification messages are sent over the session used to establish the subscription. For configured subscriptions, notification messages are sent over the connections specified by the transport and each receiver of a configured subscription.

A notification message is sent to a receiver when an event record is not blocked by either the specified filter criteria or receiver permissions. This notification message MUST be encoded in a <notification> message as defined within [RFC5277], Section 4. And per [RFC5277]'s "eventTime" object definition, the "eventTime" is populated with the event occurrence time.

The following example within [RFC7950] section 7.16.3 is an example of a compliant message:

```xml
<notification
   xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
   <eventTime>2007-09-01T10:00:00Z</eventTime>
   <link-failure xmlns="http://acme.example.com/system">
      <if-name>so-1/2/3.0</if-name>
      <if-admin-status>up</if-admin-status>
      <if-oper-status>down</if-oper-status>
   </link-failure>
</notification>
```

Figure 10: subscribed notification message

When a dynamic subscription has been started or modified, with "establish-subscription" or "modify-subscription" respectively, event records matching the newly applied filter criteria MUST NOT be sent until after the RPC reply has been sent.

When a configured subscription has been started or modified, event records matching the newly applied filter criteria MUST NOT be sent until after the "subscription-started" or "subscription-modified" notifications has been sent, respectively.
2.7. Subscription State Notifications

In addition to sending event records to receivers, a publisher MUST also send subscription state notifications when events related to subscription management have occurred.

Subscription state notifications are unlike other notifications in that they are never included in any stream. Instead, they are inserted (as defined in this section) within the sequence of notification messages sent to a particular receiver. Subscription state notifications cannot be filtered out, they cannot be stored in replay buffers, and they are delivered only to impacted receivers of a subscription. The identification of subscription state notifications is easy to separate from other notification messages through the use of the YANG extension "subscription-state-notif". This extension tags a notification as a subscription state notification.

The complete set of subscription state notifications is described in the following subsections.

2.7.1. subscription-started

This notification indicates that a configured subscription has started, and event records may be sent. Included in this state change notification are all the parameters of the subscription, except for the receiver(s) transport connection information and origin information indicating where notification messages will egress the publisher. Note that if a referenced filter from the "filters" container has been used within the subscription, the notification still provides the contents of that referenced filter under the "within-subscription" subtree.

Note that for dynamic subscriptions, no "subscription-started" notifications are ever sent.

Below is a tree diagram for "subscription-started". All objects contained in this tree are described within the included YANG model within Section 4.
2.7.2. subscription-modified

This notification indicates that a subscription has been modified by configuration operations. It is delivered directly after the last event records processed using the previous subscription parameters, and before any event records processed after the modification.

Below is a tree diagram for "subscription-modified". All objects contained in this tree are described within the included YANG model within Section 4.
A publisher most often sends this notification directly after the modification of any configuration parameters impacting a configured subscription. But it may also be sent at two other times:

1. Where a configured subscription has been modified during the suspension of a receiver, the notification will be delayed until the receiver’s suspension is lifted. In this situation, the notification indicates that the subscription has been both modified and resumed.

2. While this state change will most commonly be used with configured subscriptions, with dynamic subscriptions, there is also one time this notification will be sent. A "subscription-modified" state change notification MUST be sent if the contents of the filter identified by the subscription’s "stream-filter-ref" leaf has changed.
2.7.3. subscription-terminated

This notification indicates that no further event records for this subscription should be expected from the publisher. A publisher may terminate the sending event records to a receiver for the following reasons:

1. Configuration which removes a configured subscription, or a "kill-subscription" RPC which ends a dynamic subscription. These are identified via the reason "no-such-subscription".

2. A referenced filter is no longer accessible. This is identified by "filter-unavailable".

3. The event stream referenced by a subscription is no longer accessible by the receiver. This is identified by "stream-unavailable".

4. A suspended subscription has exceeded some timeout. This is identified by "suspension-timeout".

Each of the reasons above correspond one-to-one with a "reason" identityref specified within the YANG model.

Below is a tree diagram for "subscription-terminated". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---n subscription-terminated
    +---ro identifier    subscription-id
    +---ro reason        identityref
```

Figure 13: subscription-terminated notification tree diagram

Note: this state change notification MUST be sent to a dynamic subscription’s receiver when the subscription ends unexpectedly. The cases when this might happen are when a "kill-subscription" RPC is successful, or when some other event not including the reaching the subscription’s "stop-time" results in a publisher choosing to end the subscription.

2.7.4. subscription-suspended

This notification indicates that a publisher has suspended the sending of event records to a receiver, and also indicates the possible loss of events. Suspension happens when capacity
constraints stop a publisher from serving a valid subscription. The two conditions where this is possible are:

1. "insufficient-resources" when a publisher is unable to produce the requested event stream of notification messages, and

2. "unsupportable-volume" when the bandwidth needed to get generated notification messages to a receiver exceeds a threshold.

These conditions are encoded within the "reason" object. No further notification will be sent until the subscription resumes or is terminated.

Below is a tree diagram for "subscription-suspended". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---n subscription-suspended
    +---ro identifier   subscription-id
    +---ro reason       identityref
```

Figure 14: subscription-suspended notification tree diagram

2.7.5. subscription-resumed

This notification indicates that a previously suspended subscription has been resumed under the unmodified terms previously in place. Subscribed event records generated after the issuance of this state change notification may now be sent.

Below is the tree diagram for "subscription-resumed". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---n subscription-resumed
    +---ro identifier   subscription-id
```

Figure 15: subscription-resumed notification tree diagram

2.7.6. subscription-completed

This notification indicates that a subscription that includes a "stop-time" has successfully finished passing event records upon the reaching of that time.
Below is a tree diagram for "subscription-completed". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---n subscription-completed
    +--ro identifier    subscription-id
```

Figure 16: subscription-completed notification tree diagram

2.7.7.  replay-completed

This notification indicates that all of the event records prior to the current time have been passed to a receiver. It is sent before any notification message containing an event record with a timestamp later than (1) the "stop-time" or (2) the subscription’s start time.

If a subscription contains no "stop-time", or has a "stop-time" that has not been reached, then after the "replay-completed" notification has been sent, additional event records will be sent in sequence as they arise naturally on the publisher.

Below is a tree diagram for "replay-completed". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---n replay-completed
    +--ro identifier    subscription-id
```

Figure 17: replay-completed notification tree diagram

2.8.  Subscription Monitoring

In the operational datastore, the container "subscriptions" maintains the state of all dynamic subscriptions, as well as all configured subscriptions. Using datastore retrieval operations, or subscribing to the "subscriptions" container [I-D.ietf-netconf-yang-push] allows the state of subscriptions and their connectivity to receivers to be monitored.

Each subscription in the operational datastore is represented as a list element. Included in this list are event counters for each receiver, the state of each receiver, as well as the subscription parameters currently in effect. The appearance of the leaf "configured-subscription-state" indicates that a particular subscription came into being via configuration. This leaf also indicates if current state of that subscription is valid, invalid, and concluded.
To understand the flow of event records within a subscription, there are two counters available for each configured and dynamic receiver. The first counter is "count-sent" which shows the quantity of events actually identified for sending to a receiver. The second counter is "count-excluded" which shows event records not sent to receiver. "count-excluded" shows the combined results of both access control and per-subscription filtering. For configured subscriptions, counters are reset whenever the subscription is evaluated to valid (see (1) in Figure 8).

Dynamic subscriptions are removed from the operational datastore once they expire (reaching stop-time) or when they are terminated. While many subscription objects are shown as configurable, dynamic subscriptions are only included within the operational datastore and as a result are not configurable.

2.9. Advertisement

Publishers supporting this document MUST indicate support of the YANG model "ietf-subscribed-notifications" within the YANG library of the publisher. In addition support for optional features "encode-xml", "encode-json", "configured" "supports-vrf", "qos", "xpath", "subtree", "interface-designation", "dscp", and "replay" MUST be indicated if supported.

3. YANG Data Model Trees

This section contains tree diagrams for nodes defined in Section 4. For tree diagrams of state change notifications, see Section 2.7. Or for the tree diagrams for the RPCs, see Section 2.4.

3.1. Event Streams Container

A publisher maintains a list of available event streams as operational data. This list contains both standardized and vendor-specific event streams. This enables subscribers to discover what streams a publisher supports.

```
+--ro streams
   +--ro stream* [name]
      +--ro name                  string
      +--ro description           string
      +--ro replay-support?       empty {replay}?
      +--ro replay-log-creation-time yang:date-and-time {replay}?
      +--ro replay-log-aged-time? yang:date-and-time {replay}?
```

Figure 18: Stream Container tree diagram
Above is a tree diagram for the streams container. All objects contained in this tree are described within the included YANG model within Section 4.

3.2. Filters Container

The "filters" container maintains a list of all subscription filters that persist outside the life-cycle of a single subscription. This enables pre-defined filters which may be referenced by more than one subscription.

```
  +--rw filters
    +--rw stream-filter* [identifier]
      +--rw identifier filter-id
      +--rw (filter-spec)?
        +--:(stream-subtree-filter)
          +--rw stream-subtree-filter? <anydata> {subtree}?  
          +--:(stream-xpath-filter)
            +--rw stream-xpath-filter? yang:xpath1.0 {xpath}?  
```

Figure 19: Filter Container tree diagram

Above is a tree diagram for the filters container. All objects contained in this tree are described within the included YANG model within Section 4.

3.3. Subscriptions Container

The "subscriptions" container maintains a list of all subscriptions on a publisher, both configured and dynamic. It can be used to retrieve information about the subscriptions which a publisher is serving.

```
  +--rw subscriptions
    +--rw subscription* [identifier]
      +--rw identifier subscription-id
      +--rw (target)
        +--:(stream)
          +--rw (stream-filter)?
            +--:(by-reference)
              +--rw stream-filter-ref stream-filter-ref
              +--:(within-subscription)
                +--rw (filter-spec)?
                  +--:(stream-subtree-filter)
                    +--rw stream-subtree-filter? <anydata> {subtree}?  
```

Figure 20: Subscriptions tree diagram
Above is a tree diagram for the subscriptions container. All objects contained in this tree are described within the included YANG model within Section 4.

4. Data Model

This module imports typedefs from [RFC6991], [RFC8343], and [RFC8040], and it references [I-D.draft-ietf-rtgwg-ni-model], [XPATH], [RFC6241], [RFC7540], [RFC7951] and [RFC7950].

[ note to the RFC Editor - please replace XXXX within this YANG model with the number of this document, and XXXY with the number of [I-D.draft-ietf-rtgwg-ni-model] ]

[ note to the RFC Editor - please replace the two dates within the YANG module with the date of publication ]

<CODE BEGINS> file "ietf-subscribed-notifications@2018-08-03.yang"
module ietf-subscribed-notifications {
  yang-version 1.1;
  namespace

  prefix sn;

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-interfaces {
    prefix if;
    reference
      "RFC 8343: A YANG Data Model for Interface Management";
  }

  import ietf-network-instance {
    prefix ni;
    reference
      "draft-ietf-rtgwg-ni-model-12: YANG Model for Network Instances";
  }

  import ietf-restconf {
    prefix rc;
    reference
      "RFC 8040: RESTCONF Protocol";
  }

  import ietf-yang-types {
    prefix yang;
    reference

"RFC 6991: Common YANG Data Types";
}

organization "IETF NETCONF (Network Configuration) Working Group";
contact
"WG Web: <http:/tools.ietf.org/wg/netconf/>
WG List: <mailto:netconf@ietf.org>
Author: Alexander Clemm
<mailto:ludwig@clemm.org>
Author: Eric Voit
<mailto:evoit@cisco.com>
Author: Alberto Gonzalez Prieto
<mailto:agonzalezpri@vmware.com>
Author: Einar Nilsen-Nygaard
<mailto:einarimm@cisco.com>
Author: Ambika Prasad Tripathy
<mailto:ambtripa@cisco.com>";

description
"Contains a YANG specification for subscribing to event records and receiving matching content within notification messages.

Copyright (c) 2018 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 (https://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 2018-08-03 {
  description
    "Initial version";
  reference
    "RFC XXXX: Customized Subscriptions to a Publisher’s Event Streams";
}

/*
 * FEATURES

feature configured {
    description
    "This feature indicates that configuration of subscription is supported.";
}

feature dscp {
    description
    "This feature indicates a publisher supports the placement of suggested prioritization levels for network transport within notification messages.";
}

feature encode-json {
    description
    "This feature indicates that JSON encoding of notification messages is supported.";
}

feature encode-xml {
    description
    "This feature indicates that XML encoding of notification messages is supported.";
}

feature interface-designation {
    description
    "This feature indicates a publisher supports sourcing all receiver interactions for a configured subscription from a single designated egress interface.";
}

feature qos {
    description
    "This feature indicates a publisher supports absolute dependencies of one subscription’s traffic over another, as well as weighted bandwidth sharing between subscriptions. Both of these are Quality of Service (QoS) features which allow differentiated treatment of notification messages between a publisher and a specific receiver.";
}

feature replay {
    description
    "This feature indicates that historical event record replay is supported. With replay, it is possible for past event records to
be streamed in chronological order.
}

feature subtree {
  description
    "This feature indicates support for YANG subtree filtering.";
  reference "RFC 6241, Section 6.";
}

feature supports-vrf {
  description
    "This feature indicates a publisher supports VRF configuration
    for configured subscriptions. VRF support for dynamic
    subscriptions does not require this feature.";
  reference "RFC XXXY, Section 6.";
}

feature xpath {
  description
    "This feature indicates support for xpath filtering.";
  reference "http://www.w3.org/TR/1999/REC-xpath-19991116";
}

extension subscription-state-notification {
  description
    "This statement applies only to notifications. It indicates that
    the notification is a subscription state notification. Therefore
    it does not participate in a regular event stream and does not
    need to be specifically subscribed to in order to be received.
    This statement can only occur as a substatement to the YANG
    'notification' statement. This statement is not for use outside
    of this YANG module.";
}

identity delete-subscription-error {
  description
    "Problem found while attempting to fulfill either a
    'delete-subscription' RPC request or a 'kill-subscription'
RPC request.
}

description

"Problem found while attempting to fulfill an 'establish-subscription' RPC request."
}

identity modify-subscription-error {

description

"Problem found while attempting to fulfill a 'modify-subscription' RPC request."
}

identity subscription-suspended-reason {

description

"Problem condition communicated to a receiver as part of a 'subscription-terminated' notification."
}

identity subscription-terminated-reason {

description

"Problem condition communicated to a receiver as part of a 'subscription-terminated' notification."
}

identity dscp-unavailable {

base establish-subscription-error;

if-feature "dscp";
description

"The publisher is unable mark notification messages with a prioritization information in a way which will be respected during network transit."
}

identity encoding-unsupported {

base establish-subscription-error;
description

"Unable to encode notification messages in the desired format."
}

identity filter-unavailable {

base subscription-terminated-reason;
description

"Referenced filter does not exist. This means a receiver is referencing a filter which doesn’t exist, or to which they do not have access permissions.";
identity filter-unsupported {
  base establish-subscription-error;
  base modify-subscription-error;
  description
    "Cannot parse syntax within the filter. This failure can be from a syntax error, or a syntax too complex to be processed by the publisher.";
}

identity history-unavailable {
  base establish-subscription-error;
  if-feature "replay";
  description
    "Replay request too far into the past. This means the publisher does store historic information for the requested stream, but not back to the requested timestamp.";
}

identity insufficient-resources {
  base establish-subscription-error;
  base modify-subscription-error;
  base subscription-suspended-reason;
  description
    "The publisher has insufficient resources to support the requested subscription. An example might be that allocated CPU is too limited to generate the desired set of notification messages.";
}

identity no-such-subscription {
  base modify-subscription-error;
  base delete-subscription-error;
  base subscription-terminated-reason;
  description
    "Referenced subscription doesn’t exist. This may be as a result of a non-existent subscription ID, an ID which belongs to another subscriber, or an ID for configured subscription.";
}

identity replay-unsupported {
  base establish-subscription-error;
  if-feature "replay";
  description
    "Replay cannot be performed for this subscription. This means the publisher will not provide the requested historic information from the event stream via replay to this receiver.";
}
identity stream-unavailable {
    base subscription-terminated-reason;
    description
    "Not a subscribable stream. This means the referenced event stream
    is not available for subscription by the receiver."
}

identity suspension-timeout {
    base subscription-terminated-reason;
    description
    "Termination of previously suspended subscription. The publisher
    has eliminated the subscription as it exceeded a time limit for
    suspension."
}

identity unsupportable-volume {
    base subscription-suspended-reason;
    description
    "The publisher does not have the network bandwidth needed to get
    the volume of generated information intended for a receiver."
}

/* Identities for encodings */

identity configurable-encoding {
    description
    "If a transport identity derives from this identity, it means
    that it supports configurable encodings."
}

identity encoding {
    description
    "Base identity to represent data encodings"
}

identity encode-xml {
    base encoding;
    if-feature "encode-xml"
    description
    "Encode data using XML as described in RFC 7950"
    reference
    "RFC 7950 - The YANG 1.1 Data Modeling Language"
}

identity encode-json {
    base encoding;
if-feature "encode-json";
  description
    "Encode data using JSON as described in RFC 7951";
  reference
    "RFC 7951 - JSON Encoding of Data Modeled with YANG";
}

/* Identities for transports */
identity transport {
  description
    "An identity that represents the underlying mechanism for
    passing notification messages.";
}

identity inline-address {
  description
    "A transport identity can derive from this identity in order to
    allow inline definition of the host address in the
    'receiver' list";
}

/*
 * TYPEDEFs
 */
typedef encoding {
  type identityref {
    base encoding;
  }
  description
    "Specifies a data encoding, e.g. for a data subscription.";
}

typedef filter-id {
  type string;
  description
    "A type to identify filters which can be associated with a
    subscription.";
}

typedef stream-filter-ref {
  type leafref {
    path "/*/sn:filters/sn:stream-filter/sn:identifier";
  }
  description
    "This type is used to reference an event stream filter.";
}
typedef stream-ref {
    type leafref {
        path "*/sn:streams/sn:stream/sn:name";
    } description
        "This type is used to reference a system-provided stream.";
}

typedef subscription-id {
    type uint32;
    description
        "A type for subscription identifiers.";
}

typedef transport {
    type identityref {
        base transport;
    } description
        "Specifies transport used to send notification messages to a receiver.";
}

/*
 * GROUPINGS
 */

grouping stream-filter-elements {
    description
        "This grouping defines the base for filters applied to event streams.";
    choice filter-spec {
        description
            "The content filter specification for this request.";
        anydata stream-subtree-filter {
            if-feature "subtree";
            description
                "Event stream evaluation criteria encoded in the syntax of a subtree filter as defined in RFC 6241, Section 6.

The subtree filter is applied to the representation of individual, delineated event records as contained within the event stream. For example, if the notification message contains an instance of a notification defined in YANG, then the top-level element is the name of the YANG notification.

If the subtree filter returns a non-empty node set, the filter matches the event record, and the event record is included in
the notification message sent to the receivers."; reference "RFC 6241, Section 6.";
}
leaf stream-xpath-filter {
  if-feature "xpath";
  type yang:xpath1.0;
  description
  "Event stream evaluation criteria encoded in the syntax of an XPath 1.0 expression.

  The XPath expression is evaluated on the representation of individual, delineated event records as contained within the event stream. For example, if the notification message contains an instance of a notification defined in YANG, then the top-level element is the name of the YANG notification, and the root node has this top-level element as the only child.

  The result of the XPath expression is converted to a boolean value using the standard XPath 1.0 rules. If the boolean value is ‘true’, the filter matches the event record, and the event record is included in the notification message sent to the receivers.

  The expression is evaluated in the following XPath context:

  o The set of namespace declarations are those in scope on the 'stream-xpath-filter' leaf element.

  o The set of variable bindings is empty.

  o The function library is the core function library, and the XPath functions defined in section 10 in RFC 7950.

  o The context node is the root node."; reference "http://www.w3.org/TR/1999/REC-xpath-19991116 RFC 7950, Section 10.";
}
}

grouping update-qos {
  description
  "This grouping describes Quality of Service information concerning a subscription. This information is passed to lower layers for transport prioritization and treatment";
}
leaf dscp {
  if-feature "dscp";
  type inet:dscp;
  default "0";
  description
      "The desired network transport priority level. This is the
       priority set on notification messages encapsulating the results
       of the subscription. This transport priority is shared for all
       receivers of a given subscription.";
}
leaf weighting {
  if-feature "qos";
  type uint8 {
    range "0 .. 255";
  }
  description
      "Relative weighting for a subscription. Allows an underlying
       transport layer perform informed load balance allocations
       between various subscriptions";
  reference
      "RFC-7540, section 5.3.2";
}
leaf dependency {
  if-feature "qos";
  type subscription-id;
  description
      "Provides the 'subscription-id' of a parent subscription which
       has absolute precedence should that parent have push updates
       ready to egress the publisher. In other words, there should be
       no streaming of objects from the current subscription if
       the parent has something ready to push.

       If a dependency is asserted via configuration or via RPC, but
       the referenced 'subscription-id' does not exist, the dependency
       is silently discarded. If a referenced subscription is deleted
       this dependency is removed.";
  reference
      "RFC-7540, section 5.3.1";
}
}

grouping subscription-policy-modifiable {
  description
      "This grouping describes all objects which may be changed
       in a subscription.";
  choice target {
    mandatory true;
    description

"Identifies the source of information against which a subscription is being applied, as well as specifics on the subset of information desired from that source."

case stream {
    choice stream-filter {
        description
        "An event stream filter can be applied to a subscription. That filter will come either referenced from a global list, or be provided within the subscription itself."
        case by-reference {
            description
            "Apply a filter that has been configured separately."
            leaf stream-filter-ref {
                type stream-filter-ref;
                mandatory true;
                description
                "References an existing stream filter which is to be applied to an event stream for the subscription.";
            }
        }
        case within-subscription {
            description
            "Local definition allows a filter to have the same lifecycle as the subscription."
            uses stream-filter-elements;
        }
    }
}

leaf stop-time {
    type yang:date-and-time;
    description
    "Identifies a time after which notification messages for a subscription should not be sent. If 'stop-time' is not present, the notification messages will continue until the subscription is terminated. If 'replay-start-time' exists, 'stop-time' must be for a subsequent time. If 'replay-start-time' doesn’t exist, 'stop-time' when established must be for a future time.";
}

grouping subscription-policy-dynamic {
    description
    "This grouping describes the only information concerning a subscription which can be passed over the RPCs defined in this model."
    uses subscription-policy-modifiable {
        augment target/stream {
            ...
description
  "Adds additional objects which can be modified by RPC.";
leaf stream {
  type stream-ref {
    require-instance false;
  }
  mandatory true;
  description
    "Indicates the event stream to be considered for
     this subscription.";
}
leaf replay-start-time {
  if-feature "replay";
  type yang:date-and-time;
  config false;
  description
    "Used to trigger the replay feature for a dynamic
subscription, with event records being selected needing to
be at or after the start at the time specified. If
'replay-start-time' is not present, this is not a replay
subscription and event record push should start immediately.
It is never valid to specify start times that are later than
or equal to the current time.";
}
}
uses update-qos;
}

grouping subscription-policy {
  description
    "This grouping describes the full set of policy information
concerning both dynamic and configured subscriptions, with the
exclusion of both receivers and networking information specific to
the publisher such as what interface should be used to transmit
notification messages.";
  uses subscription-policy-dynamic;
leaf transport {
  if-feature "configured";
  type transport;
  mandatory true;
  description
    "This leaf specifies the transport used to deliver
messages destined to all receivers of a subscription.";
}
leaf encoding {
  when 'not(../transport) or derived-from(../transport,
  "sn:configurable-encoding")';
type encoding;
description
"The type of encoding for notification messages. For a
dynamic subscription, if not included as part of an establish-
subscription RPC, the encoding will be populated with the
encoding used by that RPC. For a configured subscription, if
not explicitly configured the encoding will be the default
encoding for an underlying transport."
}
leaf purpose {
  if-feature "configured";
type string;
description
"Open text allowing a configuring entity to embed the
originator or other specifics of this subscription."
}

/*
 * RPCs
 */

rpc establish-subscription {
description
"This RPC allows a subscriber to create (and possibly negotiate)
a subscription on its own behalf. If successful, the
subscription remains in effect for the duration of the
subscriber’s association with the publisher, or until the
subscription is terminated. In case an error occurs, or the
publisher cannot meet the terms of a subscription, an RPC error
is returned, the subscription is not created. In that case, the
RPC reply’s ‘error-info’ MAY include suggested parameter settings
that would have a higher likelihood of succeeding in a subsequent
‘establish-subscription’ request."
input {
  uses subscription-policy-dynamic;
  leaf encoding {
    type encoding;
description
    "The type of encoding for the subscribed data. If not
    included as part of the RPC, the encoding MUST be set by the
    publisher to be the encoding used by this RPC."
  }
}
output {
  leaf identifier {
    type subscription-id;
    mandatory true;
  }
description
"Identifier used for this subscription."
}
leaf replay-start-time-revision {
  if-feature "replay";
  type yang:date-and-time;
  description
  "If a replay has been requested, this represents the earliest time covered by the event buffer for the requested stream. The value of this object is the 'replay-log-aged-time' if it exists. Otherwise it is the 'replay-log-creation-time'. All buffered event records after this time will be replayed to a receiver. This object will only be sent if the starting time has been revised to be later than the time requested by the subscriber."
}
}

rc:yang-data establish-subscription-stream-error-info {
  container establish-subscription-stream-error-info {
    description
    "If any 'establish-subscription' RPC parameters are unsupportable against the event stream, a subscription is not created and the RPC error response MUST indicate the reason why the subscription failed to be created. This yang-data MAY be inserted as structured data within a subscription's RPC error response to indicate the failure reason. This yang-data MUST be inserted if hints are to be provided back to the subscriber."
    leaf reason {
      type identityref {
        base establish-subscription-error;
      }
      description
      "Indicates the reason why the subscription has failed to be created to a targeted stream."
    }
    leaf filter-failure-hint {
      type string;
      description
      "Information describing where and/or why a provided filter was unsupportable for a subscription."
    }
  }
}

rpc modify-subscription {
description
"This RPC allows a subscriber to modify a dynamic subscription’s parameters. If successful, the changed subscription parameters remain in effect for the duration of the subscription, until the subscription is again modified, or until the subscription is terminated. In case of an error or an inability to meet the modified parameters, the subscription is not modified and the original subscription parameters remain in effect. In that case, the RPC error MAY include ‘error-info’ suggested parameter hints that would have a high likelihood of succeeding in a subsequent ‘modify-subscription’ request. A successful ‘modify-subscription’ will return a suspended subscription to an ‘active’ state."

input {  
leaf identifier {  
  type subscription-id;  
  mandatory true;  
  description
    "Identifier to use for this subscription.";
  }  
uses subscription-policy-modifiable;
}

rc:yang-data modify-subscription-stream-error-info {  
container modify-subscription-stream-error-info {  
  description
    "This yang-data MAY be provided as part of a subscription’s RPC error response when there is a failure of a ‘modify-subscription’ RPC which has been made against a stream. This yang-data MUST be used if hints are to be provided back to the subscriber.";
  leaf reason {  
    type identityref {  
      base modify-subscription-error;
    }  
    description
      "Information in a ‘modify-subscription’ RPC error response which indicates the reason why the subscription to an event stream has failed to be modified.";
  }  
leaf filter-failure-hint {  
  type string;  
  description
    "Information describing where and/or why a provided filter was unsupportable for a subscription.";
  }
}
rpc delete-subscription {

description
"This RPC allows a subscriber to delete a subscription that
was previously created from by that same subscriber using the
‘establish-subscription’ RPC.

If an error occurs, the server replies with an ‘rpc-error’ where
the ‘error-info’ field MAY contain an
‘delete-subscription-error-info’ structure.”;

input {
leaf identifier {

type subscription-id;
mandatory true;
description
"Identifier of the subscription that is to be deleted.
Only subscriptions that were created using
‘establish-subscription’ from the same origin as this RPC
can be deleted via this RPC.”;
}
}

rpc kill-subscription {

description
"This RPC allows an operator to delete a dynamic subscription
without restrictions on the originating subscriber or underlying
transport session.

If an error occurs, the server replies with an ‘rpc-error’ where
the ‘error-info’ field MAY contain an
‘delete-subscription-error-info’ structure.”;

input {
leaf identifier {

type subscription-id;
mandatory true;
description
"Identifier of the subscription that is to be deleted. Only
subscriptions that were created using
‘establish-subscription’ can be deleted via this RPC.”;
}
}

rc:yang-data delete-subscription-error-info {

container delete-subscription-error-info {

description
"If a 'delete-subscription' RPC or a 'kill-subscription' RPC fails, the subscription is not deleted and the RPC error response MUST indicate the reason for this failure. This yang-data MAY be inserted as structured data within a subscription's RPC error response to indicate the failure reason."

leaf reason {
  type identityref {
    base delete-subscription-error;
  }
  mandatory true;
  description
    "Indicates the reason why the subscription has failed to be deleted.";
}

notification replay-completed {
  sn:subscription-state-notification;
  if-feature "replay";
  description
    "This notification is sent to indicate that all of the replay notifications have been sent. It must not be sent for any other reason."
  leaf identifier {
    type subscription-id;
    mandatory true;
    description
      "This references the affected subscription.";
  }
}

notification subscription-completed {
  sn:subscription-state-notification;
  if-feature "configured";
  description
    "This notification is sent to indicate that a subscription has finished passing event records, as the 'stop-time' has been reached."
  leaf identifier {
    type subscription-id;
    mandatory true;
    description
      "This references the affected subscription.";
  }
}
"This references the gracefully completed subscription.";
}
}
/notification subscription-modified {
    sn:subscription-state-notification;
    description
        "This notification indicates that a subscription has been modified. Notification messages sent from this point on will conform to the modified terms of the subscription. For completeness, this state change notification includes both modified and non-modified aspects of a subscription."
    leaf identifier {
        type subscription-id;
        mandatory true;
        description
            "This references the affected subscription.";
    }
}

uses subscription-policy {
    refine "target/stream/stream-filter/within-subscription" {
        description
            "Filter applied to the subscription. If the 'stream-filter-ref' is populated, the filter within the subscription came from the 'filters' container. Otherwise it is populated in-line as part of the subscription.";
    }
}

/notification subscription-resumed {
    sn:subscription-state-notification;
    description
        "This notification indicates that a subscription that had previously been suspended has resumed. Notifications will once again be sent. In addition, a 'subscription-resumed' indicates that no modification of parameters has occurred since the last time event records have been sent."
    leaf identifier {
        type subscription-id;
        mandatory true;
        description
            "This references the affected subscription.";
    }
}

/notification subscription-started {
    sn:subscription-state-notification;
    if-feature "configured";

This notification indicates that a subscription has started and notifications are beginning to be sent. This notification shall only be sent to receivers of a subscription; it does not constitute a general-purpose notification.

leaf identifier {
  type subscription-id;
  mandatory true;
  description
  "This references the affected subscription."
}

uses subscription-policy {
  refine "target/stream/replay-start-time" {
    description
    "Indicates the time that a replay using for the streaming of buffered event records. This will be populated with the most recent of the following: ‘replay-log-creation-time’, ‘replay-log-aged-time’, or the most recent publisher boot time.";
  }
  refine "target/stream/stream-filter/within-subscription" {
    description
    "Filter applied to the subscription. If the ‘stream-filter-ref’ is populated, the filter within the subscription came from the ‘filters’ container. Otherwise it is populated in-line as part of the subscription."
  }
  augment "target/stream" {
    description
    "This augmentation adds additional parameters specific to a subscription-started notification."
    leaf replay-previous-event-time {
      when "../replay-start-time";
      if-feature "replay";
      type yang:date-and-time;
      description
      "If there is at least one event in the replay buffer prior to ‘replay-start-time’, this gives the time of the event generated immediately prior to the ‘replay-start-time’. If a receiver previously received event records for this configured subscription, it can compare this time to the last event record previously received. If the two are not the same (perhaps due to a reboot), then a dynamic replay can be initiated to acquire any missing event records."
    }
  }
}
notification subscription-suspended {
    sn:subscription-state-notification;
    description
        "This notification indicates that a suspension of the subscription by the publisher has occurred. No further notifications will be sent until the subscription resumes. This notification shall only be sent to receivers of a subscription; it does not constitute a general-purpose notification."
    leaf identifier {
        type subscription-id;
        mandatory true;
        description
            "This references the affected subscription."
    }
    leaf reason {
        type identityref {
            base subscription-suspended-reason;
        }
        mandatory true;
        description
            "Identifies the condition which resulted in the suspension."
    }
}

notification subscription-terminated {
    sn:subscription-state-notification;
    description
        "This notification indicates that a subscription has been terminated."
    leaf identifier {
        type subscription-id;
        mandatory true;
        description
            "This references the affected subscription."
    }
    leaf reason {
        type identityref {
            base subscription-terminated-reason;
        }
        mandatory true;
        description
            "Identifies the condition which resulted in the termination."
    }
}
container streams {
  config false;
  description
      "This container contains information on the built-in streams
      provided by the publisher."
  list stream {
    key "name";
    description
      "Identifies the built-in event streams that are supported by the
      publisher.";
    leaf name {
      type string;
      description
      "A handle for a system-provided event stream made up of a
      sequential set of event records, each of which is
      characterized by its own domain and semantics.";
    }
    leaf description {
      type string;
      mandatory true;
      description
      "A description of the event stream, including such information
      as the type of event records that are available within this
      event stream.";
    }
    leaf replay-support {
      if-feature "replay";
      type empty;
      description
      "Indicates that event record replay is available on this
      stream.";
    }
    leaf replay-log-creation-time {
      when "./.replay-support";
      if-feature "replay";
      type yang:date-and-time;
      mandatory true;
      description
      "The timestamp of the creation of the log used to support the
      replay function on this stream. This time might be earlier
      than the earliest available information contained in the log.
      This object is updated if the log resets for some reason.";
    }
    leaf replay-log-aged-time {

if-feature "replay";
type yang:date-and-time;
description
  "The timestamp associated with last event record which has been aged out of the log. This timestamp identifies how far back into history this replay log extends, if it doesn’t extend back to the 'replay-log-creation-time'. This object MUST be present if replay is supported and any event records have been aged out of the log.";
}
}
}

container filters {
  description
  "This container contains a list of configurable filters that can be applied to subscriptions. This facilitates the reuse of complex filters once defined.";
  list stream-filter {
    key "identifier";
    description
      "A list of pre-configured filters that can be applied to subscriptions.";
    leaf identifier {
      type filter-id;
      description
        "An identifier to differentiate between filters.";
    } uses stream-filter-elements;
  }
}

container subscriptions {
  description
    "Contains the list of currently active subscriptions, i.e. subscriptions that are currently in effect, used for subscription management and monitoring purposes. This includes subscriptions that have been setup via RPC primitives as well as subscriptions that have been established via configuration.";
  list subscription {
    key "identifier";
    description
      "The identity and specific parameters of a subscription. Subscriptions within this list can be created using a control channel or RPC, or be established through configuration.

      If configuration operations or the 'kill-subscription' RPC are used to delete a subscription, a 'subscription-terminated'
message is sent to any active or suspended receivers.

leaf identifier {
  type subscription-id;
  description
    "Identifier of a subscription; unique within a publisher";
}

uses subscription-policy {
  refine "target/stream/stream" {
    description
      "Indicates the event stream to be considered for this
      subscription. If an event stream has been removed,
      and no longer can be referenced by an active subscription,
      send a 'subscription-terminated' notification with
      'stream-unavailable' as the reason. If a configured
      subscription refers to a non-existent stream, move that
      subscription to the 'invalid' state."
  }

  augment "target/stream" {
    description
      "Enables objects to added to a configured stream
      subscription";

    leaf configured-replay {
      if-feature "configured";
      if-feature "replay";
      type empty;
      description
        "The presence of this leaf indicates that replay for the
        configured subscription should start at the earliest time
        in the event log, or at the publisher boot time, which
        ever is later.";
    }
  }
}

choice notification-message-origin {
  if-feature "configured";
  description
    "Identifies the egress interface on the publisher from which
    notification messages are to be sent.";

  case interface-originated {
    description
      "When notification messages to egress a specific, designated
      interface on the publisher.";

    leaf source-interface {
      if-feature "interface-designation";
      type if:interface-ref;
      description
        "References the interface for notification messages.";
    }
  }
}
case address-originated {
  description
  "When notification messages are to depart from a publisher
  using specific originating address and/or routing context
  information.";
  leaf source-vrf {
    if-feature "supports-vrf";
    type leafref {
      path "/ni:network-instances/ni:network-instance/ni:name";
    }
    description
    "VRF from which notification messages should egress a
    publisher.";
  }
  leaf source-address {
    type inet:ip-address-no-zone;
    description
    "The source address for the notification messages. If a
    source VRF exists, but this object doesn’t, a publisher’s
    default address for that VRF must be used.";
  }
}
leaf configured-subscription-state {
  if-feature "configured";
  type enumeration {
    enum valid {
      value 1;
      description
      "Connection is active and healthy.";
    }
    enum invalid {
      value 2;
      description
      "The subscription as a whole is unsupportable with its
      current parameters.";
    }
    enum concluded {
      value 3;
      description
      "A subscription is inactive as it has hit a stop time,
      but not yet been removed from configuration.";
    }
  }
  config false;
  description
  "The presence of this leaf indicates that the subscription
originated from configuration, not through a control channel or RPC. The value indicates the system established state of the subscription.

} container receivers {
    description
    "Set of receivers in a subscription.";
    list receiver {
        key "name";
        min-elements 1;
        description
        "A host intended as a recipient for the notification messages of a subscription. For configured subscriptions, transport specific network parameters (or a leafref to those parameters) may augmented to a specific receiver within this list.";
        leaf name {
            type string;
            description
            "Identifies a unique receiver for a subscription.";
        }
        leaf count-sent {
            type yang:zero-based-counter64;
            config false;
            description
            "The number of event records sent to the receiver. The count is initialized when a dynamic subscription is established, or when a configured subscription transitions to the valid state.";
        }
        leaf count-excluded {
            type yang:zero-based-counter64;
            config false;
            description
            "The number of event records explicitly removed either via an event stream filter or an access control filter so that they are not passed to a receiver. This count is set to zero each time 'count-sent' is initialized.";
        }
        leaf state {
            type enumeration {
                enum active {
                    value 1;
                    description
                    "Receiver is currently being sent any applicable notification messages for the subscription.";
                }
                enum suspended {
                    value 2;
                    description
                    "Receiver is not currently being sent any applicable notification messages for the subscription.";
                }
            }
        }
    }
}
value 2;
description
"Receiver state is ‘suspended’, so the publisher
is currently unable to provide notification messages
for the subscription."
}

enum connecting {
  value 3;
  if-feature "configured";
description
"A subscription has been configured, but a
‘subscription-started’ state change notification needs
to be successfully received before notification
messages are sent.

If the ‘reset’ action is invoked for a receiver of an
active configured subscription, the state must be
moved to ‘connecting’.");
}

enum timeout {
  value 4;
  if-feature "configured";
description
"A subscription has failed in sending a subscription
started state change to the receiver.
Additional attempts at connection attempts are not
currently being made.";
}

config false;
mandatory true;
description
"Specifies the state of a subscription from the
perspective of a particular receiver. With this info it
is possible to determine whether a subscriber is currently
generating notification messages intended for that
receiver."
}

action reset {
  if-feature "configured";
description
"Allows the reset of this configured subscription receiver
to the ‘connecting’ state. This enables the
connection process to be re-initiated.";
output {
  leaf time {
    type yang:date-and-time;
    mandatory true;
  }
}
description
  "Time a publisher returned the receiver to a 'connecting' state."
};

choice transport {
  if-feature "configured";
  mandatory true;
  description
  "Defines the transport-specific configuration data for the value of the leaf 'transport' specified under '/subscriptions/subscription'. Individual transport specifications MUST augment this choice with YANG case statements. Each transport specific case augmentation enables the inclusion to transport parameters such as the IP address, port, and security credentials. It is these parameters which are used as necessary for making a secure transport connection to the receiver."
};

5. Considerations

5.1. IANA Considerations

This document registers the following namespace URI in the "IETF XML Registry" [RFC3688]:

Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

This document registers the following YANG module in the "YANG Module Names" registry [RFC6020]:

Name: ietf-subscribed-notifications
Prefix: sn
Reference: draft-ietf-netconf-ietf-subscribed-notifications-11.txt
 (RFC form)
5.2. Implementation Considerations

To support deployments including both configured and dynamic subscriptions, it is recommended to split subscription identifiers into static and dynamic halves. That way it eliminates the possibility of collisions if the configured subscriptions attempt to set a subscription-id which might have already been dynamically allocated. A best practice is to use lower half the "identifier" object's integer space when that "identifier" is assigned by an external entity (such as with a configured subscription). This leaves the upper half of subscription identifiers available to be dynamically assigned by the publisher.

If a subscription is unable to marshal a series of filtered event records into transmittable notification messages, the receiver should be suspended with the reason "unsupportable-volume".

For configured subscriptions, operations are against the set of receivers using the subscription identifier as a handle for that set. But for streaming updates, state change notifications are local to a receiver. In this specification it is the case that receivers get no information from the publisher about the existence of other receivers. But if a network operator wants to let the receivers correlate results, it is useful to use the subscription identifier across the receivers to allow that correlation.

For configured replay subscriptions, the receiver is protected from duplicated events being pushed after a publisher is rebooted. However it is possible that a receiver might want to acquire event records which failed to be delivered just prior to the reboot. Delivering these event records be accomplished by leveraging the "eventTime" from the last event record received prior to the receipt of a "subscription-started" state change notification. With this "eventTime" and the "replay-start-time" from the "subscription-started" notification, an independent dynamic subscription can be established which retrieves any event records which may have been generated but not sent to the receiver.

5.3. Transport Requirements

This section provides requirements for any subscribed notification transport supporting the solution presented in this document.

For both configured and dynamic subscriptions the publisher MUST authenticate a receiver via some transport level mechanism before sending any event records for which they are authorized to see. In addition, the receiver MUST authenticate the publisher at the
transport level. The result is mutual authentication between the two.

A secure transport is highly recommended and the publisher MUST ensure that the receiver has sufficient authorization to perform the function they are requesting against the specific subset of content involved.

A specific transport specification built upon this document may or may not choose to require the use of the same logical channel for the RPCs and the event records. However the event records and the subscription state notifications MUST be sent on the same transport session to ensure the properly ordered delivery.

Additional transport requirements will be dictated by the choice of transport used with a subscription. For an example of such requirements with NETCONF transport, see [I-D.draft-ietf-netconf-netconf-event-notifications].

5.4. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management transports such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The NETCONF Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF operations and content.

One subscription identifier can be used for two or more receivers of the same configured subscription. But due to the possibility of different access control permissions per receiver, it cannot be assumed that each receiver is getting identical updates.

With configured subscriptions, one or more publishers could be used to overwhelm a receiver. Notification messages SHOULD NOT be sent to any receiver which does not support this specification. Receivers that do not want notification messages need only terminate or refuse any transport sessions from the publisher.

When a receiver of a configured subscription gets a new "subscription-started" message for a known subscription where it is already consuming events, the receiver SHOULD retrieve any event...
records generated since the last event record was received. This can be
accomplish by establishing a separate dynamic replay subscription
with the same filtering criteria with the publisher", assuming the
publisher supports the "replay" feature.

For dynamic subscriptions, implementations need to protect against
malicious or buggy subscribers which may send a large number
"establish-subscription" requests, thereby using up system resources.
To cover this possibility operators SHOULD monitor for such cases
and, if discovered, take remedial action to limit the resources used,
such as suspending or terminating a subset of the subscriptions or,
if the underlying transport is session based, terminate the
underlying transport session.

There are a number of data nodes defined in this YANG module that are
writable/creatable/deletable (i.e., config true, which is the
default). These data nodes may be considered sensitive or vulnerable
in some network environments. Write operations (e.g., edit-config)
to these data nodes without proper protection can have a negative
effect on network operations. These are the subtrees and data nodes
where there is a specific sensitivity/vulnerability:

Container: "/filters"
  o "stream-subtree-filter": updating a filter could increase the
    computational complexity of all referencing subscriptions.
  o "stream-xpath-filter": updating a filter could increase the
    computational complexity of all referencing subscriptions.

Container: "/subscriptions"

The following considerations are only relevant for configuration
operations made upon configured subscriptions:
  o "configured-replay": can be used to send a large number of event
    records to a receiver.
  o "dependency": can be used to force important traffic to be queued
    behind less important updates.
  o "dscp": if unvalidated, can result in the sending of traffic with
    a higher priority marking than warranted.
  o "identifier": can overwrite an existing subscription, perhaps one
    configured by another entity.
- "name": adding a new key entry can be used to attempt to send traffic to an unwilling receiver.
- "replay-start-time": can be used to push very large logs, wasting resources.
- "source-address": the configured address might not be able to reach a desired receiver.
- "source-interface": the configured interface might not be able to reach a desired receiver.
- "source-vrf": can place a subscription into a virtual network where receivers are not entitled to view the subscribed content.
- "stop-time": could be used to terminate content at an inopportune time.
- "stream": could set a subscription to an event stream containing no content permitted for the targeted receivers.
- "stream-filter-ref": could be set to a filter which is irrelevant to the event stream.
- "stream-subtree-filter": a complex filter can increase the computational resources for this subscription.
- "stream-xpath-filter": a complex filter can increase the computational resources for this subscription.
- "transport": this YANG choice node can be augmented with transport parameters which could then send the subscribed information to an undesired receiver.
- "weighting": placing a large weight can overwhelm the dequeuing of other subscriptions.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

Container: "/streams"

- "name": if access control is not properly configured, can expose system internals to those who should have no access to this information.
o "replay-support": if access control is not properly configured, can expose logs to those who should have no access.

Container: "/subscriptions"

o "count-excluded": leaf can provide information about filtered event records. A network operator should have permissions to know about such filtering.

o "subscription": different operational teams might have a desire to set varying subsets of subscriptions. Access control should be designed to permit read access to just the allowed set.

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

RPC: all

o If a malicious or buggy subscriber sends an unexpectedly large number of RPCs, the result might be an excessive use of system resources on the publisher just to determine that these subscriptions should be declined. In such a situation, subscription interactions MAY be terminated by terminating the transport session.

RPC: "delete-subscription"

o No special considerations.

RPC: "establish-subscription"

o Subscriptions could overload a publisher’s resources. For this reason, publishers MUST ensure that they have sufficient resources to fulfill this request or otherwise reject the request.

RPC: "kill-subscription"

o The "kill-subscription" RPC MUST be secured so that only connections with administrative rights are able to invoke this RPC.

RPC: "modify-subscription"

o Subscriptions could overload a publisher’s resources. For this reason, publishers MUST ensure that they have sufficient resources to fulfill this request or otherwise reject the request.
6. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Andy Bierman, Tim Jenkins, Martin Bjorklund, Kent Watsen, Balazs Lengyel, Robert Wilton, Sharon Chisholm, Hector Trevino, Susan Hares, Michael Scharf, and Guangying Zheng.

7. References

7.1. Normative References

[I-D.draft-ietf-rtgwg-ni-model]


7.2. Informative References

[I-D.draft-ietf-netconf-netconf-client-server]
Watsen, K. and G. Wu, "NETCONF Client and Server Models",
draft-ietf-netconf-netconf-client-server-06 (work in
progress), June 2018.

[I-D.draft-ietf-netconf-netconf-event-notifications]
Clemm, Alexander., Voit, Eric., Gonzalez Prieto, Alberto.,
Nilsen-Nygaard, E., and A. Tripathy, "NETCONF support for
event notifications", May 2018,
<https://datatracker.ietf.org/doc/
draft-ietf-netconf-netconf-event-notifications/>.

[I-D.draft-ietf-netconf-restconf-notif]
Voit, Eric., Clemm, Alexander., Tripathy, A., Nilsen-
Nygaard, E., and Alberto. Gonzalez Prieto, "Restconf and
HTTP transport for event notifications", May 2018,
<https://datatracker.ietf.org/doc/
draft-ietf-netconf-restconf-notif/>.

[I-D.ietf-netconf-yang-push]
Clemm, Alexander., Voit, Eric., Gonzalez Prieto, Alberto.,
Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B.
Lengyel, "YANG Datastore Subscription", May 2018,
<https://datatracker.ietf.org/doc/
draft-ietf-netconf-yang-push/>.

Transfer Protocol Version 2 (HTTP/2)", RFC 7540,
DOI 10.17487/RFC7540, May 2015,

for Subscription to YANG Datastores", RFC 7923,
DOI 10.17487/RFC7923, June 2016,

[RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home",
RFC 8071, DOI 10.17487/RFC8071, February 2017,

BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018,
Appendix A. Example Configured Transport Augmentation

This appendix provides a non-normative example of how the YANG model defined in Section 4 may be enhanced to incorporate the configuration parameters needed to support the transport connectivity process. In this example, NETCONF transport connectivity is explored. Other transports may be supported via a similar YANG model. For more on the overall need, see Section 2.5.7.

Within the YANG model defined below are two main elements. First is a transport identity "netconf". This transport identity allows a configuration agent to define NETCONF as the selected type of transport for a subscription. Second is a YANG case augmentation "netconf" which is made to the 
"/subscriptions/subscription/receivers/receiver/transport" choice node of Section 4. Within this augmentation are the necessary transport configuration parameters. In this case the parameters consist of a leafref to the ietf-netconf-client.yang model of [I-D.draft-ietf-netconf-netconf-client-server]. This leafref itself allows referencing to the actual parameters which would be used to establish the transport connection.

<CODE BEGINS> file
"ietf-netconf-subscribed-notifications@2018-08-03.yang"
module ietf-netconf-subscribed-notifications {
  yang-version 1.1;
  namespace
  prefix nsn;

  import ietf-subscribed-notifications {
    prefix sn;
  }
  import ietf-netconf-client {
    prefix ncc;
  }

  organization "IETF NETCONF (Network Configuration) Working Group";
  contact
    "WG Web: <http://tools.ietf.org/wg/netconf/>
    WG List: <mailto:netconf@ietf.org>");
  description
    "Defines NETCONF as a supported transport for subscribed event
    notifications.";
  revision 2018-08-03 {

description
    "Initial version";
reference
    "RFC XXXX: Customized Subscriptions to a Publisher’s Event Streams";
}

identity netconf {
    base sn:transport;
    base sn:inline-address;
    description
        "NETCONF is used as a configured subscription transport for
        notification messages and state change notifications."
    reference
        "RFC-6241: Network Configuration Protocol (NETCONF)";
}

augment "/sn:subscriptions/sn:subscription/sn:receivers/sn:receiver" + "/sn:transport" {
    when 'derived-from(../../../../transport, "nsn:netconf")';
    description
        "This augmentation allows NETCONF specific parameters to be
        exposed for a receiver.";
    case "netconf" {
        description
            "This case allows NETCONF specific parameters to be
            exposed for a receiver.";
        leaf netconf-endpoint {
            type leafref {
                path "/ncc:netconf-client/ncc:initiate/ncc:netconf-server" + 
                "/ncc:endpoints/ncc:endpoint/ncc:name";
            }
            mandatory true;
            description
                "Remote client which need to initiate the NETCONF transport if
                an existing NETCONF session from that client is not
                available.";
        }
    }
}

<CODE ENDS>

Figure 21: Example Transport Augmentation for NETCONF
Appendix B. Changes between revisions

(To be removed by RFC editor prior to publication)

v14 - v15
  o Text tweaks.
  o Mandatory empty case "transport" added for transport parameters.
    This includes a new section and an appendix explaining it.

v13 - v14
  o Removed the 'address' leaf.
  o Replay is now of type 'empty' for configured.

v12 - v13
  o Tweaks from Kent's comments
  o Referenced in YANG model updated per Tom Petch's comments
  o Added leaf replay-previous-event-time
  o Renamed the event counters, downshifted the subscription states

v11 - v12
  o Tweaks from Kent's, Tim's, and Martin's comments
  o Clarified dscp text, and made its own feature
  o YANG model tweaks alphabetizing, features.

v10 - v11
  o access control filtering of events in streams included to match
    RFC5277 behavior
  o security considerations updated based on YANG template.
  o dependency QoS made non-normative on HTTP2 QoS
  o tree diagrams referenced for each figure using them
  o reference numbers placed into state machine figures
broke configured replay into its own section

many tweaks updates based on LC and YANG doctor reviews

trees and YANG model reconciled were deltas existed

new feature for interface originated.

dscp removed from the qos feature

YANG model updated in a way which collapses groups only used once
so that they are part of the ‘subscriptions’ container.

alternative encodings only allowed for transports which support
them.

v09 - v10

Typos and tweaks

v08 - v09

NMDA model supported. Non NMDA version at
https://github.com/netconf-wg/rfc5277bis/

Error mechanism revamped to match to embedded implementations.

Explicitly identified error codes relevant to each RPC/
Notification

v07 - v08

Split YANG trees to separate document subsections.

Clarified configured state machine based on Balazs comments, and
moved it into the configured subscription subsections.

Normative reference to Network Instance model for VRF

One transport for all receivers of configured subscriptions.

QoS section moved in from yang-push

v06 - v07

Clarification on state machine for configured subscriptions.

v05 - v06
o Made changes proposed by Martin, Kent, and others on the list. Most significant of these are stream returned to string (with the SYSLOG identity removed), intro section on 5277 relationship, an identity set moved to an enumeration, clean up of definitions/terminology, state machine proposed for configured subscriptions with a clean-up of subscription state options.

o JSON and XML become features. Also Xpath and subtree filtering become features

o Terminology updates with event records, and refinement of filters to just event stream filters.

o Encoding refined in establish-subscription so it takes the RPC’s encoding as the default.

o Namespaces in examples fixed.

v04 – v05

o Returned to the explicit filter subtyping of v00

o stream object changed to ‘name’ from ‘stream’

o Cleaned up examples

o Clarified that JSON support needs notification-messages draft.

v03 – v04

o Moved back to the use of RFC5277 one-way notifications and encodings.

v03 – v04

o Replay updated

v02 – v03

o RPCs and Notification support is identified by the Notification 2.0 capability.

o Updates to filtering identities and text

o New error type for unsupportable volume of updates

o Text tweaks.
v01 - v02

- Subscription status moved under receiver.

v00 - v01

- Security considerations updated
- Intro rewrite, as well as scattered text changes
- Added Appendix A, to help match this to related drafts in progress
- Updated filtering definitions, and filter types in yang file, and moved to identities for filter types
- Added Syslog as an event stream
- HTTP2 moved in from YANG-Push as a transport option
- Replay made an optional feature for events. Won’t apply to datastores
- Enabled notification timestamp to have different formats.
- Two error codes added.

v01 5277bis - v00 subscribed notifications

- Kill subscription RPC added.
- Renamed from 5277bis to Subscribed Notifications.
- Changed the notification capabilities version from 1.1 to 2.0.
- Extracted create-subscription and other elements of RFC5277.
- Error conditions added, and made specific in return codes.
- Simplified yang model structure for removal of ‘basic’ grouping.
- Added a grouping for items which cannot be statically configured.
- Operational counters per receiver.
- Subscription-id and filter-id renamed to identifier
- Section for replay added. Replay now cannot be configured.
Control plane notification renamed to subscription state notification

Source address: Source-vrf changed to string, default address option added

In yang model: ‘info’ changed to ‘policy’

Scattered text clarifications

YANG Model changes. New groupings for subscription info to allow restriction of what is changeable via RPC. Removed notifications for adding and removing receivers of configured subscriptions.

Expanded/renamed definitions from event server to publisher, and client to subscriber as applicable. Updated the definitions to include and expand on RFC 5277.

Removal of redundancy with other drafts

Many other clean-ups of wording and terminology

Authors’ Addresses

Eric Voit
Cisco Systems
Email: evoit@cisco.com

Alexander Clemm
Huawei
Email: ludwig@clemm.org

Alberto Gonzalez Prieto
Microsoft
Email: alberto.gonzalez@microsoft.com

Einar Nilsen-Nygaard
Cisco Systems
Email: einarnn@cisco.com
Abstract

This document defines three YANG modules: the first defines groupings for a generic TLS client, the second defines groupings for a generic TLS server, and the third defines common identities and groupings used by both the client and the server. It is intended that these groupings will be used by applications using the TLS protocol.

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.

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

- I-D.ietf-netconf-trust-anchors
- I-D.ietf-netconf-keystore

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

- "XXXX" --> the assigned RFC value for this draft
- "YYYY" --> the assigned RFC value for I-D.ietf-netconf-trust-anchors
- "ZZZZ" --> the assigned RFC value for I-D.ietf-netconf-keystore

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

- "2018-06-04" --> the publication date of this draft
The following Appendix section is to be removed prior to publication:

- Appendix A. 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 December 6, 2018.

Copyright Notice

Copyright (c) 2018 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 ................................................. 3
2. Terminology .................................................. 4
3. The TLS Client Model ......................................... 4
   3.1. Tree Diagram ........................................... 4
   3.2. Example Usage .......................................... 5
   3.3. YANG Module ........................................... 7
4. The TLS Server Model .......................................... 10
   4.1. Tree Diagram ........................................... 10
   4.2. Example Usage .......................................... 11
   4.3. YANG Module ........................................... 13
1. Introduction

This document defines three YANG 1.1 [RFC7950] modules: the first defines a grouping for a generic TLS client, the second defines a grouping for a generic TLS server, and the third defines identities and groupings common to both the client and the server (TLS is defined in [RFC5246]). It is intended that these groupings will be used by applications using the TLS protocol. For instance, these groupings could be used to help define the data model for an HTTPS [RFC2818] server or a NETCONF over TLS [RFC7589] based server.

The client and server YANG modules in this document each define one grouping, which is focused on just TLS-specific configuration, and specifically avoids any transport-level configuration, such as what ports to listen-on or connect-to. This affords applications the opportunity to define their own strategy for how the underlying TCP connection is established. For instance, applications supporting NETCONF Call Home [RFC8071] could use the "ssh-server-grouping" grouping for the TLS parts it provides, while adding data nodes for the TCP-level call-home configuration.

The modules defined in this document uses groupings defined in [I-D.ietf-netconf-keystore] enabling keys to be either locally defined or a reference to globally configured values.
2. 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. The TLS Client Model

3.1. Tree Diagram

This section provides two tree diagrams [RFC8340] for the "ietf-tls-client" module, the first with used groupings expanded and the second with used groupings not expanded.

The following tree diagram has used groupings expanded:

module: ietf-tls-client

grouping tls-client-grouping
  +- client-identity
    |  +- (auth-type)?
    |     +-:(certificate)
    |        +- certificate
    |           +- (local-or-keystore)
    |               +-:(local)
    |                  +- algorithm
    |                     |     ct:key-algorithm-ref
    |                     +- public-key       binary
    |                     +- private-key     union
    |                     +- cert
    |                       |     ct:end-entity-cert-cms
    |                       +---n certificate-expiration
    |                           +- expiration-date?  yang:date-and-time
    |                          +--+:(keystore)  (keystore-implemented)?
    |                              +- reference
    |                                 ks:asymmetric-key-certificate-ref
    
    +- server-auth
    |  +- pinned-ca-certs?     ta:pinned-certificates-ref
    |  +- pinned-server-certs? ta:pinned-certificates-ref
    
    +- hello-params {tls-client-hello-params-config}?
    |  +- tls-versions
    |     +- tls-version*      identityref
    |  +- cipher-suites
    |     +- cipher-suite*     identityref

The following tree diagram does not have the groupings expanded:
3.2.  Example Usage

This section presents two examples showing the tls-client-grouping populated with some data. These examples are effectively the same except the first configures the client identity using a local key while the second uses a key configured in a keystore. Both examples are consistent with the examples presented in Section 3 of [I-D.ietf-netconf-trust-anchors] and Section 3.2 of [I-D.ietf-netconf-keystore].

The following example configures the client identity using a local key:
[Note: `\` line wrapping for formatting only]

```xml
  <!-- how this client will authenticate itself to the server -->
  <client-identity>
    <certificate>
      <private-key>base64encodedvalue==</private-key>
      <public-key>base64encodedvalue==</public-key>
      <cert>base64encodedvalue==</cert>
    </certificate>
  </client-identity>

  <!-- which certificates will this client trust -->
  <server-auth>
    <pinned-ca-certs>explicitly-trusted-server-ca-certs</pinned-ca-certs>
    <pinned-server-certs>explicitly-trusted-server-certs</pinned-server-certs>
  </server-auth>
</tls-client>
```

The following example configures the client identity using a key from the keystore:
[Note: ‘\’ line wrapping for formatting only]

  <!-- how this client will authenticate itself to the server -->
  <client-identity>
    <certificate>
      <reference>ex-rsa-cert</reference>
    </certificate>
  </client-identity>

  <!-- which certificates will this client trust -->
  <server-auth>
    <pinned-ca-certs>explicitly-trusted-server-ca-certs</pinned-ca-certs>
    <pinned-server-certs>explicitly-trusted-server-certs</pinned-server-certs>
  </server-auth>
</tls-client>

3.3. YANG Module

This YANG module has normative references to
[I-D.ietf-netconf-trust-anchors] and [I-D.ietf-netconf-keystore].

<CODE BEGINS> file "ietf-tls-client@2018-06-04.yang"
module ietf-tls-client {
  yang-version 1.1;

  prefix "tlsc";

  import ietf-tls-common {
    prefix tlscmn;
    revision-date 2018-06-04; // stable grouping definitions
    reference
      "RFC XXXX: YANG Groupings for TLS Clients and TLS Servers";
  }

  import ietf-trust-anchors {
    prefix ta;
    reference
      "RFC YYYY: YANG Data Model for Global Trust Anchors";
  }

  import ietf-keystore {
    prefix ks;
  }
</CODE ENDS>
This module defines a reusable grouping for a TLS client that can be used as a basis for specific TLS client instances.

Copyright (c) 2018 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 "2018-06-04" {
  description "Initial version";
  reference "RFC XXXX: YANG Groupings for TLS Clients and TLS Servers";
}

// features

feature tls-client-hello-params-config {
  description "TLS hello message parameters are configurable on a TLS client.";
}
grouping tls-client-grouping {
  description
  "A reusable grouping for configuring a TLS client without any consideration for how an underlying TCP session is established.";
}

container client-identity {
  description
  "The credentials used by the client to authenticate to the TLS server."
  choice auth-type {
    description
    "The authentication type."
    container certificate {
      uses ks:local-or-keystore-end-entity-certificate-grouping;
      description
      "A locally-defined or referenced certificate to be used for client authentication."
      reference
      "RFC ZZZZ: YANG Data Model for a 'Keystore' Mechanism";
    }
  }
}

container server-auth {
  must 'pinned-ca-certs or pinned-server-certs';
  description
  "Trusted server identities."
  leaf pinned-ca-certs {
    type ta:pinned-certificates-ref;
    description
    "A reference to a list of certificate authority (CA) certificates used by the TLS client to authenticate TLS server certificates. A server certificate is authenticated if it has a valid chain of trust to a configured pinned CA certificate.";
  }
  leaf pinned-server-certs {
    type ta:pinned-certificates-ref;
    description
    "A reference to a list of server certificates used by the TLS client to authenticate TLS server certificates.
"
A server certificate is authenticated if it is an exact match to a configured pinned server certificate.

\[
\text{container hello-params { if-feature tls-client-hello-params-config; uses tlscmn:hello-params-grouping; description "Configurable parameters for the TLS hello message."; }}
\]

\[
\text{)} // end tls-client-grouping}
\]

4. The TLS Server Model

4.1. Tree Diagram

This section provides two tree diagrams [RFC8340] for the "ietf-tls-server" module, the first with used groupings expanded and the second with used groupings not expanded.

The following tree diagram has used groupings expanded:
module: ietf-tls-server

grouping tls-server-grouping
  +++ server-identity
  |  +++ (local-or-keystore)
  |  |  +++:(local)
  |  |  |  +++ algorithm ct:key-algorithm-ref
  |  |  |  +++ public-key binary
  |  |  |  +++ private-key union
  |  |  +++ cert ct:end-entity-cert-cms
  |  |  |  +++-- n certificate-expiration
  |  |  |  |  +++ expiration-date? yang:date-and-time
  |  |  |  +++:(keystore) {keystore-implemented}?
  |  |  |  |  +++ reference
  |  |  |  |  |  ks:asymmetric-key-certificate-ref
  |  |  +++ client-auth
  |  |  |  +++ pinned-ca-certs? ta:pinned-certificates-ref
  |  |  |  +++ pinned-client-certs? ta:pinned-certificates-ref
  |  |  +++ hello-params {tls-server-hello-params-config}?
  |  |  |  +++ tls-versions
  |  |  |  |  +++ tls-version* identityref
  |  |  |  +++ cipher-suites
  |  |  |  |  +++ cipher-suite* identityref

The following tree diagram does not have the used groupings expanded:

module: ietf-tls-server

  grouping tls-server-grouping
  |  +++ server-identity
  |  |  +++:(local-or-keystore-end-entity-certificate-grouping)
  |  +++ client-auth
  |  |  +++ pinned-ca-certs? ta:pinned-certificates-ref
  |  |  +++ pinned-client-certs? ta:pinned-certificates-ref
  |  +++ hello-params {tls-server-hello-params-config}?
  |  |  +++ tls-versions
  |  |  |  +++ tls-version* identityref
  |  |  +++ cipher-suites
  |  |  |  +++ cipher-suite* identityref

4.2. Example Usage

This section presents two examples showing the tls-server-grouping populated with some data. These examples are effectively the same except the first configures the server identity using a local key while the second uses a key configured in a keystore. Both examples are consistent with the examples presented in Section 3 of [I-D.ietf-netconf-trust-anchors] and Section 3.2 of [I-D.ietf-netconf-keystore].
The following example configures the server identity using a local key:

[Note: '\ ' line wrapping for formatting only]

```xml
<tls-server xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-server">
  <!-- how this server will authenticate itself to the client -->
  <server-identity>
      <private-key>base64encodedvalue==</private-key>
      <public-key>base64encodedvalue==</public-key>
      <cert>base64encodedvalue==</cert>
    </algorithm>
  </server-identity>

  <!-- which certificates will this server trust -->
  <client-auth>
    <pinned-ca-certs>explicitly-trusted-client-ca-certs</pinned-ca-certs>
    <pinned-client-certs>explicitly-trusted-client-certs</pinned-client-certs>
  </client-auth>
</tls-server>
```

The following example configures the server identity using a key from the keystore:
[Note: ‘\’ line wrapping for formatting only]

<tls-server xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-server">
  <!-- how this server will authenticate itself to the client -->
  <server-identity>
    <reference>ex-rsa-cert</reference>
  </server-identity>

  <!-- which certificates will this server trust -->
  <client-auth>
    <pinned-ca-certs>explicitly-trusted-client-ca-certs</pinned-ca-certs>
    <pinned-client-certs>explicitly-trusted-client-certs</pinned-client-certs>
  </client-auth>
</tls-server>

4.3. YANG Module

This YANG module has a normative references to [RFC5246],
[I-D.ietf-netconf-trust-anchors] and [I-D.ietf-netconf-keystore].

<CODE BEGINS> file "ietf-tls-server@2018-06-04.yang"
module ietf-tls-server {
  yang-version 1.1;

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

  import ietf-tls-common {
    prefix tlscmn;
    revision-date 2018-06-04; // stable grouping definitions
    reference
      "RFC XXXX: YANG Groupings for TLS Clients and TLS Servers";
  }

  import ietf-trust-anchors {
    prefix ta;
    reference
      "RFC YYYY: YANG Data Model for Global Trust Anchors";
  }

  import ietf-keystore {
    prefix ks;
    reference
      "RFC ZZZZ: YANG Data Model for a 'Keystore' Mechanism";
  }

Watsen & Wu                  Expires December 6, 2018                  [Page 13]
This module defines a reusable grouping for a TLS server that can be used as a basis for specific TLS server instances.

Copyright (c) 2018 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 "2018-06-04" {
  description
    "Initial version";
  reference
    "RFC XXXX: YANG Groupings for TLS Clients and TLS Servers";
}

// features

feature tls-server-hello-params-config {
  description
    "TLS hello message parameters are configurable on a TLS server.";
}
// groupings

grouping tls-server-grouping {
  description
  "A reusable grouping for configuring a TLS server without any consideration for how underlying TCP sessions are established.";

  container server-identity {
    description
    "A locally-defined or referenced end-entity certificate, including any configured intermediate certificates, the TLS server will present when establishing a TLS connection in its Certificate message, as defined in Section 7.4.2 in RFC 5246.";
    reference
    RFC ZZZZ: YANG Data Model for a 'Keystore' Mechanism";
    uses ks:local-or-keystore-end-entity-certificate-grouping;
  }

  container client-auth {
    description
    "A reference to a list of pinned certificate authority (CA) certificates and a reference to a list of pinned client certificates.";
    leaf pinned-ca-certs {
      type ta:pinned-certificates-ref;
      description
      "A reference to a list of certificate authority (CA) certificates used by the TLS server to authenticate TLS client certificates. A client certificate is authenticated if it has a valid chain of trust to a configured pinned CA certificate.";
      reference
      "RFC YYYY: YANG Data Model for Global Trust Anchors";
    }
    leaf pinned-client-certs {
      type ta:pinned-certificates-ref;
      description
      "A reference to a list of client certificates used by the TLS server to authenticate TLS client certificates. A clients certificate is authenticated if it is an exact match to a configured pinned client certificate.";
      reference
      "RFC YYYY: YANG Data Model for Global Trust Anchors";
  }
}
5. The TLS Common Model

The TLS common model presented in this section contains identities and groupings common to both TLS clients and TLS servers. The hello-params-grouping can be used to configure the list of TLS algorithms permitted by the TLS client or TLS server. The lists of algorithms are ordered such that, if multiple algorithms are permitted by the client, the algorithm that appears first in its list that is also permitted by the server is used for the TLS transport layer connection. The ability to restrict the algorithms allowed is provided in this grouping for TLS clients and TLS servers that are capable of doing so and may serve to make TLS clients and TLS servers compliant with security policies.

Features are defined for algorithms that are OPTIONAL or are not widely supported by popular implementations. Note that the list of algorithms is not exhaustive.

5.1. Tree Diagram

The following tree diagram [RFC8340] provides an overview of the data model for the "ietf-tls-common" module.

```
module: ietf-tls-common

  grouping hello-params-grouping
    --- tls-versions
      |  --- tls-version* identityref
      --- cipher-suites
        --- cipher-suite* identityref
```
5.2. Example Usage

This section shows how it would appear if the transport-params-grouping were populated with some data.

```xml
  <tls-params>
    <tls-params>
      <tls-params>tls-1.1</tls-2.2>
    </tls-versions>
  </cipher-suites>
</hello-params>

5.3. YANG Module

This YANG module has a normative references to [RFC2246], [RFC4346], [RFC4492], [RFC5246], [RFC5288], and [RFC5289].

```xml
<CODE BEGINS> file "ietf-tls-common@2018-06-04.yang" module ietf-tls-common { yang-version 1.1;

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

  contact  "WG Web: <http://datatracker.ietf.org/wg/netconf/>
            WG List: <mailto:netconf@ietf.org>
            Author: Kent Watsen
                    <mailto:kwatsen@juniper.net>
            Author: Gary Wu
                    <mailto:garywu@cisco.com>

  description  "This module defines a common features, identities, and groupings
                for Transport Layer Security (TLS).";

Watsen & Wu           Expires December 6, 2018          [Page 17]"
Copyright (c) 2018 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 "2018-06-04" {
  description
    "Initial version";
  reference
    "RFC XXXX: YANG Groupings for TLS Clients and TLS Servers";
}

// features

feature tls-1_0 {
  description
    "TLS Protocol Version 1.0 is supported.";
  reference
    "RFC 2246: The TLS Protocol Version 1.0";
}

feature tls-1_1 {
  description
    "TLS Protocol Version 1.1 is supported.";
  reference
}

feature tls-1_2 {
  description
    "TLS Protocol Version 1.2 is supported.";
  reference
}

feature tls-ecc {
  description
    "Elliptic Curve Cryptography (ECC) is supported for TLS.";
}
reference
"RFC 4492: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)";

feature tls-dhe {
  description
  "Ephemeral Diffie-Hellman key exchange is supported for TLS.";
  reference
}

feature tls-3des {
  description
  "The Triple-DES block cipher is supported for TLS.";
  reference
}

feature tls-gcm {
  description
  "The Galois/Counter Mode authenticated encryption mode is supported for TLS.";
  reference
  "RFC 5288: AES Galois Counter Mode (GCM) Cipher Suites for TLS";
}

feature tls-sha2 {
  description
  "The SHA2 family of cryptographic hash functions is supported for TLS.";
  reference
  "FIPS PUB 180-4: Secure Hash Standard (SHS)";
}

// identities

identity tls-version-base {
  description
  "Base identity used to identify TLS protocol versions.";
}

identity tls-1.0 {
  base tls-version-base;
  if-feature tls-1_0;
}
description
  "TLS Protocol Version 1.0."
reference
  "RFC 2246: The TLS Protocol Version 1.0"
}

identity tls-1.1 {
  base tls-version-base;
  if-feature tls-1_1;
  description
    "TLS Protocol Version 1.1."
  reference
}

identity tls-1.2 {
  base tls-version-base;
  if-feature tls-1_2;
  description
    "TLS Protocol Version 1.2."
  reference
}

identity cipher-suite-base {
  description
    "Base identity used to identify TLS cipher suites."
}

identity rsa-with-aes-128-cbc-sha {
  base cipher-suite-base;
  description
    "Cipher suite TLS_RSA_WITH_AES_128_CBC_SHA."
  reference
}

identity rsa-with-aes-256-cbc-sha {
  base cipher-suite-base;
  description
    "Cipher suite TLS_RSA_WITH_AES_256_CBC_SHA."
  reference
}
identity rsa-with-aes-128-cbc-sha256 {
    base cipher-suite-base;
    if-feature tls-sha2;
    description "Cipher suite TLS_RSA_WITH_AES_128_CBC_SHA256.";
}

identity rsa-with-aes-256-cbc-sha256 {
    base cipher-suite-base;
    if-feature tls-sha2;
    description "Cipher suite TLS_RSA_WITH_AES_256_CBC_SHA256.";
}

identity dhe-rsa-with-aes-128-cbc-sha {
    base cipher-suite-base;
    if-feature tls-dhe;
    description "Cipher suite TLS_DHE_RSA_WITH_AES_128_CBC_SHA.";
}

identity dhe-rsa-with-aes-256-cbc-sha {
    base cipher-suite-base;
    if-feature tls-dhe;
    description "Cipher suite TLS_DHE_RSA_WITH_AES_256_CBC_SHA.";
}

identity dhe-rsa-with-aes-128-cbc-sha256 {
    base cipher-suite-base;
    if-feature "tls-dhe and tls-sha2";
    description "Cipher suite TLS_DHE_RSA_WITH_AES_128_CBC_SHA256.";
}
identity dhe-rsa-with-aes-256-cbc-sha256 {
    base cipher-suite-base;
    if-feature "tls-dhe and tls-sha2";
    description
        "Cipher suite TLS_DHE_RSA_WITH_AES_256_CBC_SHA256.";
    reference
}

identity ecdhe-ecdsa-with-aes-128-cbc-sha256 {
    base cipher-suite-base;
    if-feature "tls-ecc and tls-sha2";
    description
        "Cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256.";
    reference
        "RFC 5289: TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)";
}

identity ecdhe-ecdsa-with-aes-256-cbc-sha384 {
    base cipher-suite-base;
    if-feature "tls-ecc and tls-sha2";
    description
        "Cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384.";
    reference
        "RFC 5289: TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)";
}

identity ecdhe-rsa-with-aes-128-cbc-sha256 {
    base cipher-suite-base;
    if-feature "tls-ecc and tls-sha2";
    description
        "Cipher suite TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256.";
    reference
        "RFC 5289: TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)";
}

identity ecdhe-rsa-with-aes-256-cbc-sha384 {
    base cipher-suite-base;
    if-feature "tls-ecc and tls-sha2";
    description
        "Cipher suite TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384.";
    reference
        "RFC 5289: TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)";
}
"RFC 5289: TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)"

identity ecdhe-ecdsa-with-aes-128-gcm-sha256 {
    base cipher-suite-base;
    if-feature "tls-ecc and tls-gcm and tls-sha2";
    description
        "Cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256.";
    reference
        "RFC 5289: TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)"
}

identity ecdhe-ecdsa-with-aes-256-gcm-sha384 {
    base cipher-suite-base;
    if-feature "tls-ecc and tls-gcm and tls-sha2";
    description
        "Cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384.";
    reference
        "RFC 5289: TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)"
}

identity ecdhe-rsa-with-aes-128-gcm-sha256 {
    base cipher-suite-base;
    if-feature "tls-ecc and tls-gcm and tls-sha2";
    description
        "Cipher suite TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256.";
    reference
        "RFC 5289: TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)"
}

identity ecdhe-rsa-with-aes-256-gcm-sha384 {
    base cipher-suite-base;
    if-feature "tls-ecc and tls-gcm and tls-sha2";
    description
        "Cipher suite TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384.";
    reference
        "RFC 5289: TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)"
}

identity rsa-with-3des-ede-cbc-sha {
    base cipher-suite-base;
    if-feature tls-3des;
    description
"Cipher suite TLS_RSA_WITH_3DES_EDE_CBC_SHA.";
reference
Version 1.2";
}

identity ecdhe-rsa-with-3des-ede-cbc-sha {
  base cipher-suite-base;
  if-feature "tls-ecc and tls-3des";
  description
    "Cipher suite TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA.";
  reference
    "RFC 4492: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)";
}

identity ecdhe-rsa-with-aes-128-cbc-sha {
  base cipher-suite-base;
  if-feature "tls-ecc";
  description
    "Cipher suite TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA.";
  reference
    "RFC 4492: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)";
}

identity ecdhe-rsa-with-aes-256-cbc-sha {
  base cipher-suite-base;
  if-feature "tls-ecc";
  description
    "Cipher suite TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA.";
  reference
    "RFC 4492: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)";
}

// groupings

grouping hello-params-grouping {
  description
    "A reusable grouping for TLS hello message parameters.";
  reference
  
  container tls-versions {
    description
      "Parameters regarding TLS versions.";
  }
  
  container cipher-suites {
    description
      "Cipher suites supported by the client and server.";
  }
}

Watsen & Wu  Expires December 6, 2018  [Page 24]
leaf-list tls-version {
    type identityref {
        base tls-version-base;
    }
    description
    "Acceptable TLS protocol versions.
    If this leaf-list is not configured (has zero elements)
    the acceptable TLS protocol versions are implementation-defined.";
}
}

container cipher-suites {
    description
    "Parameters regarding cipher suites.";
    leaf-list cipher-suite {
        type identityref {
            base cipher-suite-base;
        }
        ordered-by user;
        description
        "Acceptable cipher suites in order of descending preference.
        If this leaf-list is not configured (has zero elements)
        the acceptable cipher suites are implementation-defined.";
    }
}

} // end hello-params-grouping

6. Security Considerations

The YANG modules defined in this document are designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.
Since the modules defined in this document only define groupings, these considerations are primarily for the designers of other modules that use these groupings.

There are a number of data nodes defined in the YANG modules that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

/: The entire data tree of all the groupings defined in this draft is sensitive to write operations. For instance, the addition or removal of references to keys, certificates, trusted anchors, etc., can dramatically alter the implemented security policy. However, no NACM annotations are applied as the data SHOULD be editable by users other than a designated ‘recovery session’.

Some of the readable data nodes in the YANG modules may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

NONE

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

NONE

7. IANA Considerations

7.1. The IETF XML Registry

This document registers three URIs in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:
7.2. The YANG Module Names Registry

This document registers three YANG modules in the YANG Module Names registry [RFC7950]. Following the format in [RFC7950], the following registrations are requested:

name:         ietf-tls-client
prefix:       tlsc
reference:    RFC XXXX

name:         ietf-tls-server
prefix:       tlss
reference:    RFC XXXX

name:         ietf-tls-common
prefix:       tlscmn
reference:    RFC XXXX

8. Acknowledgements

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

9. References

9.1. Normative References

Watsen, K., "YANG Data Model for Global Trust Anchors", draft-ietf-netconf-trust-anchors-00 (work in progress), June 2018.


Watsen & Wu Expires December 6, 2018 [Page 28]
9.2. Informative References


Appendix A. Change Log

A.1. 00 to 01
  o Noted that ‘0.0.0.0’ and ‘::’ might have special meanings.
  o Renamed "keychain" to "keystore".

A.2. 01 to 02
  o Removed the groupings containing transport-level configuration.
    Now modules contain only the transport-independent groupings.
  o Filled in previously incomplete ’ietf-tls-client’ module.
  o Added cipher suites for various algorithms into new ’ietf-tls-
    common’ module.

A.3. 02 to 03
  o Added a ‘must’ statement to container ‘server-auth’ asserting that
    at least one of the various auth mechanisms must be specified.
  o Fixed description statement for leaf ‘trusted-ca-certs’.

A.4. 03 to 04
  o Updated title to "YANG Groupings for TLS Clients and TLS Servers"
  o Updated leafref paths to point to new keystore path
  o Changed the YANG prefix for ietf-tls-common from ‘tlscom’ to
    ‘tlscmn’.
  o Added TLS protocol versions 1.0 and 1.1.
  o Made author lists consistent
  o Now tree diagrams reference ietf-netmod-yang-tree-diagrams
  o Updated YANG to use typedefs around leafrefs to common keystore
    paths
  o Now inlines key and certificates (no longer a leafref to keystore)
A.5. 04 to 05
   - Merged changes from co-author.

A.6. 05 to 06
   - Updated to use trust anchors from trust-anchors draft (was
     keystore draft)
   - Now Uses new keystore grouping enabling asymmetric key to be
     either locally defined or a reference to the keystore.

Authors’ Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Gary Wu
Cisco Systems

EMail: garywu@cisco.com
Abstract

This document describes a UDP-based publication channel for streaming telemetry use to collect data from devices. A new shim header is proposed to facilitate the distributed data collection mechanism which directly pushes data from line cards to the collector. Because of the lightweight UDP encapsulation, higher frequency and better transit performance can be achieved.

Requirements Language

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].

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 January 1, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

Streaming telemetry refers to sending a continuous stream of operational data from a device to a remote receiver. This provides an ability to monitor a network from remote and to provide network analytics. Devices generate telemetry data and push that data to a
collector for further analysis. By streaming the data, much better performance, finer-grained sampling, monitoring accuracy, and bandwidth utilization can be achieved than with polling-based alternatives.

Sub-Notif [I-D.ietf-netconf-subscribed-notifications] defines a mechanism that allows a collector to subscribe to updates of YANG-defined data that is maintained in a YANG [RFC7950] datastore. The mechanism separates the management and control of subscriptions from the transport that is used to actually stream and deliver the data. Two transports, NETCONF transport [I-D.ietf-netconf-netconf-event-notifications] and HTTP transport [I-D.ietf-netconf-restconf-notif], have been defined so far for the notification messages.

While powerful in its features and general in its architecture, in its current form the mechanism needs to be extended to stream telemetry data at high velocity from devices that feature a distributed architecture. The transports that have been defined so far, NETCONF and HTTP, are ultimately based on TCP and lack the efficiency needed to stream data continuously at high velocity. A lighter-weight, more efficient transport, e.g. a transport based on UDP is needed.

- Firstly, data collector will suffer a lot of TCP connections from, for example, many line cards equipped on different devices.
- Secondly, as no connection state needs to be maintained, UDP encapsulation can be easily implemented by hardware which will further improve the performance.
- Thirdly, because of the lightweight UDP encapsulation, higher frequency and better transit performance can be achieved, which is important for streaming telemetry.

This document specifies a higher-performance transport option for Sub-Notif that leverages UDP. Specifically, it facilitates the distributed data collection mechanism described in [I-D.zhou-netconf-multi-stream-originators]. In the case of data originating from multiple line cards, the centralized design requires data to be internally forwarded from those line cards to the push server, presumably on a main board, which then combines the individual data items into a single consolidated stream. The centralized data collection mechanism can result in a performance bottleneck, especially when large amounts of data are involved. What is needed instead is the support for a distributed mechanism that allows to directly push multiple individual substreams, e.g. one from each line card, without needing to first pass them through an
additional processing stage for internal consolidation, but still allowing those substreams to be managed and controlled via a single subscription. The proposed UDP based Publication Channel (UPC) natively supports the distributed data collection mechanism.

The transport described in this document can be used for transmitting notification messages over both IPv4 and IPv6 [RFC8200].

While this document will focus on the data publication channel, the subscription can be used in conjunction with the mechanism proposed in [I-D.ietf-netconf-subscribed-notifications] with extensions [I-D.zhou-netconf-multi-stream-originators].

2. Terminology

Streaming Telemetry: refers to sending a continuous stream of operational data from a device to a remote receiver. This provides an ability to monitor a network from remote and to provide network analytics.

Component Subscription: A subscription that defines the data from each individual telemetry source which is managed and controlled by a single Subscription Server.

Component Subscription Server: An agent that streams telemetry data per the terms of a component subscription.

3. Solution Overview

The typical distributed data collection solution is shown in Fig. 1. Both the Collector and the Publisher can be distributed. The Collector includes the Subscriber and a set of Receivers. And the Publisher includes a Subscription Server and a set of Component Subscription Servers. The Subscriber cannot see the Component Subscription Servers directly, so it will send the Global Subscription information to the Subscription Server (e.g., main board) via the Subscription Channel. When receiving a Global Subscription, the Subscription Server decomposes the subscription request into multiple Component Subscriptions, each involving data from a separate internal telemetry source, for example a line card. The Component Subscriptions are distributed to the Component Subscription Server. Subsequently, each data originator generates its own stream of telemetry data, collecting and encapsulating the packets per the Component Subscription and streaming them to the designated Receivers. This distributed data collection mechanism may form multiple Publication Channels to the Receivers. The Receiver is able to assemble many pieces of data associated with one Global Subscription.
The Publication Channel supports the reliable data streaming, for example for some alarm events. The Collector has the option of deducing the packet loss and the disorder based on the information carried by the notification data. And the Collector may decide the behavior to request retransmission.

The rest of the draft describes the UDP based Publication Channel (UPC).

4. Transport Mechanisms

For a complete pub-sub mechanism, this section will describe how the UPC is used to interact with the Subscription Channel relying on NETCONF or RESTCONF.

4.1. Dynamic Subscription

Dynamic subscriptions for Sub-Notif are configured and managed via signaling messages transported over NETCONF [RFC6241] or RESTCONF [RFC8040]. The Sub-Notif defined RPCs which are sent and responded via the Subscription Channel (a), between the Subscriber and the Subscription Server of the Publisher. In this case, only one
Receiver is associated with the Subscriber. In the Publisher, there may be multiple data originators. Notification messages are pushed on separate channels (b), from different data originators to the Receiver.

![Call Flow For Dynamic Subscription](image)

In the case of dynamic subscription, the Receiver and the Subscriber SHOULD be colocated. So UPC can use the source IP address of the Subscription Channel as it’s destination IP address. The Receiver MUST support listening messages at the IANA-assigned PORT-X or PORT-Y, but MAY be configured to listen at a different port.

The Publication Channels MUST share fate with the subscription session. In other words, when the delete-subscription is received or the subscription session is broken, all the associated Publication Channels MUST be closed.
4.2. Configured Subscription

For a Configured Subscription, there is no guarantee that the Subscriber is currently in place with the associated Receiver(s). As defined in Sub-Notif, the subscription configuration contains the location information of all the receivers, including the IP address and the port number. So that the data originator can actively send generated messages to the corresponding Receivers via the UPC.

The first message MUST be a separate subscription-started notification to indicate the Receiver that the pushing is started. Then, the notifications can be sent immediately without any wait.

All the subscription state notifications, as defined in [I-D.ietf-netconf-subscribed-notifications], MUST be encapsulated to be separated notification messages.

---

Fig. 3 Call Flow For Configured Subscription
5. UDP Transport for Publication Channel

5.1. Design Overview

As specified in Sub-Notif, the telemetry data is encapsulated in the NETCONF/RESTCONF notification message, which is then encapsulated and carried in the transport protocols, e.g. TLS, HTTP2. The following figure shows the overview of the typical UPC message structure.

- The Message Header contains information that can facilitate the message transmission before de-serializing the notification message.
- Notification Message is the encoded content that the publication channel transports. The common encoding method includes GPB [1], CBOR [RFC7049], JSON, and XML. [I-D.ietf-netconf-notification-messages] describes the structure of the Notification Message for both single notification and multiple bundled notifications.

```
+-------+---------------+-------+-------------------------------+
| Vers. |    Flag       |  ET   |      Length                   |
|       | Message-Generator-ID |       |                               |
+---------------------------------------------------------------+
|                       Message ID                              |
|                                                                 |
+---------------------------------------------------------------+
˜                       Options                                 ˜
+---------------------------------------------------------------+

Fig. 4 UDP Publication Message Overview
```

5.2. Data Format of the UPC Message Header

The UPC Message Header contains information that can facilitate the message transmission before de-serializing the notification message. The data format is shown as follows.

```
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
+-------+---------------+-------+-------------------------------+
| Vers. |    Flag       |  ET   |      Length                   |
+---------------------------------------------------------------+
|                       Message-Generator-ID |       |                               |
+---------------------------------------------------------------+
|                       Message ID                              |
+---------------------------------------------------------------+
-                       Options                                 -
+---------------------------------------------------------------+

Fig. 3 UPC Message Header Format
The Message Header contains the following field:

- **Vers.**: represents the PDU (Protocol Data Unit) encoding version. The initial version value is 0.

- **Flag**: is a bitmap indicating what features this packet has and the corresponding options attached. Each bit associates to one feature and one option data. When the bit is set to 1, the associated feature is enabled and the option data is attached. The sequence of the presence of the options follows the bit order of the bitmap. In this document, the flag is specified as follows:
  
  * bit 0, the reliability flag;
  * bit 1, the fragmentation flag;
  * other bits are reserved.

- **ET**: is a 4 bits identifier to indicate the encoding type used for the Notification Message. 16 types of encoding can be expressed:
  
  * 0: GPB;
  * 1: CBOR;
  * 2: JSON;
  * 3: XML;
  * others are reserved.

- **Length**: is the total length of the message, measured in octets, including message header.

- **Message-Generator-ID**: is a 32-bit identifier of the process which created the notification message. This allows disambiguation of an information source, such as the identification of different line cards sending the notification messages. The source IP address of the UDP datagrams SHOULD NOT be interpreted as the identifier for the host that originated the UPC message. The entity sending the UPC message could be merely a relay.

- The Message ID is generated continuously by the message generator. Different subscribers share the same notification ID sequence.

- **Options**: is a variable-length field. The details of the Options will be described in the respective sections below.
5.3. Options

The order of packing the data fields in the Options field follows the bit order of the Flag field.

5.3.1. Reliability Option

The UDP based publication transport described in this document provides two streaming modes, the reliable mode and the unreliable mode, for different SLA (Service Level Agreement) and telemetry requirements.

In the unreliable streaming mode, the line card pushes the encapsulated data to the data collector without any sequence information. So the subscriber does not know whether the data is correctly received or not. Hence no retransmission happens.

The reliable streaming mode provides sequence information in the UDP packet, based on which the subscriber can deduce the packet loss and disorder. Then the subscriber can decide whether to request the retransmission of the lost packets.

In most case, the unreliable streaming mode is preferred. Because the reliable streaming mode will cost more network bandwidth and precious device resource. Different from the unreliable streaming mode, the line card cannot remove the sent reliable notifications immediately, but to keep them in the memory for a while. Reliable notifications may be pushed multiple times, which will increase the traffic. When choosing the reliable streaming mode or the unreliable streaming mode, the operate need to consider the reliable requirement together with the resource usage.

When the reliability flag bit is set to 1 in the Flag field, the following option data will be attached

\[
\begin{array}{cccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 & 1 \\
|---------------------|---------------------|---------------------|
Previous Message ID
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
</tbody>
</table>
\end{array}
\]

Fig. 4 Reliability Option Format

Current Message ID and Previous Message ID will be added in the packets.

For example, there are two subscriber A and B,
Message IDs for the generator are: [1, 2, 3, 4, 5, 6, 7, 8, 9], in which Subscriber A subscribes [1, 2, 3, 6, 7] and Subscriber B subscribes [1, 2, 4, 5, 7, 8, 9].

Subscriber A will receive [Previous Message ID, Current Message ID] like: [0,1][1,2][2,3][3,6][6,7].

Subscriber B will receive [Previous Message ID, Current Message ID] like: [0,1][1,2][2,4][4,5][5,7][7,8][8,9].

5.3.2. Fragmentation Option

UDP payload has a theoretical length limitation to 65535. Other encapsulation headers will make the actual payload even shorter. Binary encodings like GPB and CBOR can make the message compact. So that the message can be encapsulated within one UDP packet, hence fragmentation will not easily happen. However, text encodings like JSON and XML can easily make the message exceed the UDP length limitation.

The Fragmentation Option can help not Application layer can split the YANG tree into several leaves. Or table into several rows. But the leaf or the row cannot be split any further. Now we consider a very long path. Since the GPB and CBOR are so compact, it’s easy to fit into a UDP packet. But for JSON or XML, it is possible that even one leaf will exceed the UDP boundary.

```
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
+-------------------------------------------------------------+-+
|            Fagment Number                                    |L|
+-------------------------------------------------------------+-+
```

Fig. 5 Fragmentation Option Format

The Fragmentation Option is available in the message header when the fragmentation flag is set to 1. The option contains:

Fragment Number: indicates the sequence number of the current fragment.

L: is a flag to indicate whether the current fragment is the last one. When 0 is set, current fragment is not the last one, hence more fragments are expected. When 1 is set, current fragment is the last one.
5.4. Data Encoding

Subscribed data can be encoded in GPB, CBOR, XML or JSON format. It is conceivable that additional encodings may be supported as options in the future. This can be accomplished by augmenting the subscription data model with additional identity statements used to refer to requested encodings.

Implementation may support different encoding method per subscription. When bundled notifications is supported between the publisher and the receiver, only subscribed notifications with the same encoding can be bundled as one message.

6. Using DTLS to Secure UPC

The Datagram Transport Layer Security (DTLS) protocol [RFC6347] is designed to meet the requirements of applications that need secure datagram transport.

DTLS can be used as a secure transport to counter all the primary threats to UDP based Publication Channel:

- Confidentiality to counter disclosure of the message contents.
- Integrity checking to counter modifications to a message on a hop-by-hop basis.
- Server or mutual authentication to counter masquerade.

In addition, DTLS also provides:

- A cookie exchange mechanism during handshake to counter Denial of Service attacks.
- A sequence number in the header to counter replay attacks.

6.1. Transport

As shown in Figure 6, the DTLS is layered next to the UDP transport to provide reusable security and authentication functions over UDP. No DTLS extension is required to enable UPC messages over DTLS.
The application implementer will map a unique combination of the remote address, remote port number, local address, and local port number to a session.

Each UPC message is delivered by the DTLS record protocol, which assigns a sequence number to each DTLS record. Although the DTLS implementer may adopt a queue mechanism to resolve reordering, it may not assure that all the messages are delivered in order when mapping on the UDP transport.

Since UDP is an unreliable transport, with DTLS, an originator or relay may not realize that a collector has gone down or lost its DTLS connection state, so messages may be lost.

The DTLS record has its own sequence number, the encryption and decryption will done by DTLS layer, UPC Message layer will not concern this.

6.2. Port Assignment

The Publisher is always a DTLS client, and the Receiver is always a DTLS server. The Receivers MUST support accepting UPC Messages on the UDP port PORT-Y, but MAY be configurable to listen on a different port. The Publisher MUST support sending UPC messages to the UDP port PORT-Y, but MAY be configurable to send messages to a different port. The Publisher MAY use any source UDP port for transmitting messages.

6.3. DTLS Session Initiation

The Publisher initiates a DTLS connection by sending a DTLS Client Hello to the Receiver. Implementations MUST support the denial of service countermeasures defined by DTLS. When these countermeasures are used, the Receiver responds with a DTLS Hello Verify Request containing a cookie. The Publisher responds with a DTLS Client Hello containing the received cookie, which initiates the DTLS handshake.
The Publisher MUST NOT send any UPC messages before the DTLS handshake has successfully completed.

Implementations MUST support DTLS 1.0 [RFC4347] and MUST support the mandatory to implement cipher suite, which is TLS_RSA_WITH_AES_128_CBC_SHA [RFC5246] as specified in DTLS 1.0. If additional cipher suites are supported, then implementations MUST NOT negotiate a cipher suite that employs NULL integrity or authentication algorithms.

Where privacy is REQUIRED, then implementations must either negotiate a cipher suite that employs a non-NULL encryption algorithm or else achieve privacy by other means, such as a physically secured network.

6.4. Sending Data

All UPC messages MUST be sent as DTLS "application data". It is possible that multiple UPC messages be contained in one DTLS record, or that a publication message be transferred in multiple DTLS records. The application data is defined with the following ABNF [RFC5234] expression:

```
APPLICATION-DATA = 1*UPC-FRAME
UPC-FRAME = MSG-LEN SP UPC-MSG
MSG-LEN = NONZERO-DIGIT *DIGIT
SP = %d32
NONZERO-DIGIT = %d49-57
DIGIT = %d48 / NONZERO-DIGIT
```

UPC-MSG is defined in section 5.2.

6.5. Closure

A Publisher MUST close the associated DTLS connection if the connection is not expected to deliver any UPC Messages later. It MUST send a DTLS close_notify alert before closing the connection. A Publisher (DTLS client) MAY choose to not wait for the Receiver's close_notify alert and simply close the DTLS connection. Once the Receiver gets a close_notify from the Publisher, it MUST reply with a close_notify.

When no data is received from a DTLS connection for a long time (where the application decides what "long" means), Receiver MAY close
the connection. The Receiver (DTLS server) MUST attempt to initiate
an exchange of close_notify alerts with the Publisher before closing
the connection. Receivers that are unprepared to receive any more
data MAY close the connection after sending the close_notify alert.

Although closure alerts are a component of TLS and so of DTLS, they,
like all alerts, are not retransmitted by DTLS and so may be lost
over an unreliable network.

7. Congestion Control

Congestion control mechanisms that respond to congestion by reducing
traffic rates and establish a degree of fairness between flows that
share the same path are vital to the stable operation of the Internet
[RFC2914]. While efficient, UDP has no build-in congestion control
mechanism. Because streaming telemetry can generate unlimited
amounts of data, transferring this data over UDP is generally
problematic. It is not recommended to use the UDP based publication
channel over congestion-sensitive network paths. The only
environments where the UDP based publication channel MAY be used are
managed networks. The deployments require the network path has been
explicitly provisioned for the UDP based publication channel through
traffic engineering mechanisms, such as rate limiting or capacity
reservations.

8. IANA Considerations

This RFC requests that IANA assigns three UDP port numbers in the
"Registered Port Numbers" range with the service names "upc" and
"upc-dtls". These ports will be the default ports for the UDP based
Publication Channel for NETCONF and RESTCONF. Below is the
registration template following the rules in [RFC6335].

Service Name: upc
Transport Protocol(s): UDP
Assignee: IESG <iesg@ietf.org>
Contact: IETF Chair <chair@ietf.org>
Description: UDP based Publication Channel
Reference: RFC XXXX
Port Number: PORT-X
Service Name: upc-dtls
Transport Protocol(s): UDP
Assignee: IESG <iesg@ietf.org>
Contact: IETF Chair <chair@ietf.org>
Description: UDP based Publication Channel (DTLS)
Reference: RFC XXXX
Port Number: PORT-Y

9. Security Considerations

TBD

10. Acknowledgements

The authors of this documents would like to thank Eric Voit, Tim Jenkins, and Huiyang Yang for the initial comments.

11. References

11.1. Normative References


11.2. Informative References

[I-D.ietf-netconf-netconf-event-notifications]

[I-D.ietf-netconf-notification-messages]
11.3. URIs


Appendix A. Change Log

(To be removed by RFC editor prior to publication)

A.1. draft-ietf-zheng-udp-pub-channel-00 to v00

- Modified the message header format.
- Added a section on the Authentication Option.
- Cleared up the text and removed unnecessary TBDs.

A.2. v01

- Removed the detailed description on distributed data collection mechanism from this document. Mainly focused on the description of a UDP-based publication channel for telemetry use.
- Modified the message header format.

A.2. v02

- Add the section on the transport mechanism.
- Modified the fixed message header format.
o Add the fragmentation option for the message header.

A.2. v03

o Clarify term through the document.

o Add a section on DTLS support.

Authors’ Addresses

Guangying Zheng
Huawei
101 Yu-Hua-Tai Software Road
Nanjing, Jiangsu
China

Email: zhengguangying@huawei.com

Tianran Zhou
Huawei
156 Beiqing Rd., Haidian District
Beijing
China

Email: zhoutianran@huawei.com

Alexander Clemm
Huawei
2330 Central Expressway
Santa Clara, California
USA

Email: alexander.clemm@huawei.com
YANG Datastore Subscription
draft-ietf-netconf-yang-push-17

Abstract

Via the mechanism described in this document, subscriber applications may request a continuous, customized stream of updates from a YANG datastore. Providing such visibility into changes made upon YANG configuration and operational objects enables new capabilities based on the remote mirroring of configuration and operational state.

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 January 2, 2019.
Copyright Notice

Copyright (c) 2018 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.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction ................................................. 3
2. Definitions and Acronyms ................................. 4
3. Solution Overview .......................................... 5
   3.1. Subscription Model ................................... 5
   3.2. Negotiation of Subscription Policies .................... 6
   3.3. On-Change Considerations ........................... 7
   3.4. Reliability Considerations ........................ 8
   3.5. Data Encodings ...................................... 9
   3.6. Defining the Selection with a Datastore ............... 10
   3.7. Streaming Updates .................................. 11
   3.8. Subscription Management ........................... 13
   3.9. Receiver Authorization ............................ 14
   3.10. On-change Notifiable YANG objects .................. 15
   3.11. Other Considerations ............................ 16
4. A YANG data model for management of datastore push
   subscriptions ........................................... 17
   4.1. Overview ........................................... 17
   4.2. Subscription configuration ........................... 22
   4.3. YANG Notifications ................................. 23
1. Introduction

Traditional approaches to remote visibility have been built on polling. With polling, data is periodically requested and retrieved by a client from a server to stay up-to-date. However, there are issues associated with polling-based management:

- Polling incurs significant latency. This latency prohibits many application types.
- Polling cycles may be missed, requests may be delayed or get lost, often when the network is under stress and the need for the data is the greatest.
- Polling requests may undergo slight fluctuations, resulting in intervals of different lengths. The resulting data is difficult to calibrate and compare.
- For applications that monitor for changes, many remote polling cycles place ultimately fruitless load on the network, devices, and applications.

A more effective alternative to polling is for an application to receive automatic and continuous updates from a targeted subset of a datastore. Accordingly, there is a need for a service that allows applications to subscribe to updates from a datastore and that enables the publisher to push and in effect stream those updates. The requirements for such a service have been documented in [RFC7923].

This document defines a corresponding solution that is built on top of "Custom Subscription to Event Streams" [I-D.draft-ietf-netconf-subscribed-notifications]. Supplementing that work are YANG data model augmentations, extended RPCs, and new
datastore specific update notifications. Transport options for [I-D.draft-ietf-netconf-subscribed-notifications] will work seamlessly with this solution.

2. Definitions and Acronyms

This document uses the terminology defined in [RFC7950], [RFC8341], and [RFC8342]. In addition, the following terms are introduced:

- **Datastore node**: An instance of management information in a datastore. Also known as "object".

- **Datastore node update**: A data item containing the current value of a datastore node at the time the datastore node update was created.

- **Datastore subscription**: A subscription to updates regarding contents of a datastore.

- **Datastore subtree**: An instantiated datastore node and the datastore nodes that are hierarchically contained within it.

- **On-change subscription**: A datastore subscription with updates that are triggered when changes in subscribed datastore nodes are detected.

- **Periodic subscription**: A datastore subscription with updates that are triggered periodically according to some time interval.

- **Selection filter**: Evaluation and/or selection criteria, which may be applied against a targeted set of objects.

- **Update record**: A representation of one or more datastore node updates. In addition, an update record may contain which type of update led to the datastore node update (e.g., whether the datastore node was added, changed, deleted). Also included in the update record may be other metadata, such as a subscription identifier of the subscription as part of which the update record was generated.

- **Update trigger**: A mechanism that determines when an update record needs to be generated.

- **YANG-Push**: The subscription and push mechanism for datastore updates that is specified in this document.
3. Solution Overview

This document specifies a solution that provides subscription service for updates from a datastore. This solution supports dynamic as well as configured subscriptions to updates of datastore nodes. Subscriptions specify when notification messages should be sent and what data to include in update records. YANG objects are subsequently pushed from the publisher to the receiver per the terms of the subscription.

3.1. Subscription Model

YANG-push subscriptions are defined using a data model that is itself defined in YANG. This model enhances the subscription model defined in [I-D.draft-ietf-netconf-subscribed-notifications] with capabilities that allow subscribers to subscribe to datastore node updates, specifically to specify the update triggers defining when to generate update records as well as what to include in an update record. Key enhancements include:

- Specification of selection filters which identify targeted YANG datastore nodes and/or subtrees within a datastore for which updates are to be pushed.

- Specification of update policies contain conditions which trigger the generation and pushing of new update records. There are two types of subscriptions, distinguished by how updates are triggered: periodic and on-change.

  * For periodic subscriptions, the update trigger is specified by two parameters that define when updates are to be pushed. These parameters are the period interval with which to report updates, and an anchor time which can be used to calculate at which point in time updates need to be assembled and sent.

  * For on-change subscriptions, an update trigger occurs whenever a change in the subscribed information is detected. Included are additional parameters such as:

    + Dampening period: In an on-change subscription, detected object changes should be sent as quickly as possible. However it may be undesirable to send a rapid series of object changes. Such behavior has the potential to exhaust resources in the publisher or receiver. In order to protect against that, a dampening period MAY be used to specify the interval which must pass before successive update records for the same subscription are generated for a receiver. The dampening period collectively applies to the
set of all datastore nodes selected by a single subscription and sent to a single receiver. This means that when there is a change to one or more subscribed objects, an update record containing those objects is created either immediately when no dampening period is in effect, or at the end of a dampening period. If multiple changes to a single object occur during a dampening period, only the value that is in effect at the time the update record is created is included. The dampening period goes into effect every time an update record completes assembly.

+ Change type: This parameter can be used to reduce the types of datastore changes for which updates are sent (e.g., you might only send when an object is created or deleted, but not when an object value changes).

+ No Synch on start: defines whether or not a complete push-update of all subscribed data will be sent at the beginning of a subscription. Such early synchronization establishes the frame of reference for subsequent updates.

3.2. Negotiation of Subscription Policies

A dynamic subscription request SHOULD be declined if a publisher’s assessment is that it may be unable to provide update records meeting the terms of an "establish-subscription" or "modify-subscription" rpc request. In this case, a subscriber may quickly follow up with a new rpc request using different parameters.

Random guessing at different parameters by a subscriber is to be discouraged. Therefore, in order to minimize the number of subscription iterations between subscriber and publisher, dynamic subscription supports a simple negotiation between subscribers and publishers for subscription parameters. This negotiation is in the form of supplemental information which may be inserted within error responses to a failed rpc request. This returned error response information, when considered, should increase the likelihood of success for subsequent rpc requests. Such hints include suggested periodic time intervals, acceptable dampening periods, and size estimates for the number or objects which would be returned from a proposed selection filter. However, there are no guarantees that subsequent requests which consider these hints will be accepted.
3.3. On-Change Considerations

On-change subscriptions allow subscribers to receive updates whenever changes to targeted objects occur. As such, on-change subscriptions are particularly effective for data that changes infrequently, yet for which applications need to be quickly notified whenever a change does occur with minimal delay.

On-change subscriptions tend to be more difficult to implement than periodic subscriptions. Accordingly, on-change subscriptions may not be supported by all implementations or for every object.

Whether or not to accept or reject on-change subscription requests when the scope of the subscription contains objects for which on-change is not supported is up to the publisher implementation. A publisher MAY accept an on-change subscription even when the scope of the subscription contains objects for which on-change is not supported. In that case, updates are sent only for those objects within the scope that do support on-change updates whereas other objects are excluded from update records, whether or not their values actually change. In order for a subscriber to determine whether objects support on-change subscriptions, objects are marked accordingly on a publisher. Accordingly, when subscribing, it is the responsibility of the subscriber to ensure it is aware of which objects support on-change and which do not. For more on how objects are so marked, see Section 3.10.

Alternatively, a publisher MAY decide to simply reject an on-change subscription in case the scope of the subscription contains objects for which on-change is not supported. In case of a configured subscription, the subscription MAY be suspended.

To avoid flooding receivers with repeated updates for subscriptions containing fast-changing objects, or objects with oscillating values, an on-change subscription allows for the definition of a dampening period. Once an update record for a given object is generated, no other updates for this particular subscription will be created until the end of the dampening period. Values sent at the end of the dampening period are the current values of all changed objects which are current at the time the dampening period expires. Changed objects include those which were deleted or newly created during that dampening period. If an object has returned to its original value (or even has been created and then deleted) during the dampening period, the last change will still be sent. This will indicate churn is occurring on that object.

On-change subscriptions can be refined to let users subscribe only to certain types of changes. For example, a subscriber might only want
object creations and deletions, but not modifications of object values.

Putting it all together, following is the conceptual process for creating an push-change-update notification:

1. Just before a change, or at the start of a dampening period, evaluate any filtering and any access control rules. The result is a set "A" of datastore nodes and subtrees.

2. Just after a change, or at the end of a dampening period, evaluate any filtering and any (possibly new) access control rules. The result is a set "B" of datastore nodes and subtrees.

3. Construct a YANG patch record for going from A to B.

4. If there were any changes made between A and B which canceled each other out, insert into the YANG patch record the last change made for any object which otherwise wouldn’t have appeared.

5. If the resulting patch record is non-empty, send it to the receiver.

Note: In cases where a subscriber wants to have separate dampening periods for different objects, multiple subscriptions with different objects in a selection filter can be created.

3.4. Reliability Considerations

A subscription to updates from a datastore is intended to obviate the need for polling. However, in order to do so, it is critical that subscribers can rely on the subscription and have confidence that they will indeed receive the subscribed updates without having to worry about updates being silently dropped. In other words, a subscription constitutes a promise on the side of the publisher to provide the receivers with updates per the terms of the subscription.

Now, there are many reasons why a publisher may at some point no longer be able to fulfill the terms of the subscription, even if the subscription had been entered into with good faith. For example, the volume of data objects may be larger than anticipated, the interval may prove too short to send full updates in rapid succession, or an internal problem may prevent objects from being collected. For this reason, the solution that is defined in this document mandates that a publisher notifies receivers immediately and reliably whenever it encounters a situation in which it is unable to keep the terms of the subscription, and provides the publisher with the option to suspend the subscription in such a case.
A publisher SHOULD reject a request for a subscription if it is unlikely that the publisher will be able fulfill the terms of that subscription request. In such cases, it is preferable to have a subscriber request a less resource intensive subscription than to deal with frequently degraded behavior.

3.5. Data Encodings

3.5.1. Periodic Subscriptions

In a periodic subscription, the data included as part of an update corresponds to data that could have been read using a retrieval operation.

3.5.2. On-Change Subscriptions

In an on-change subscription, updates need to indicate not only values of changed datastore nodes but also the types of changes that occurred since the last update. Therefore encoding rules for data in on-change updates will generally follow YANG-patch operation as specified in [RFC8072]. The YANG-patch will describe what needs to be applied to the earlier state reported by the preceding update, to result in the now-current state. Note that contrary to [RFC8072], objects encapsulated are not restricted to configuration objects only.

A publisher will indicate a change to the effect that a value of a datastore node has been updated by indicating a "replace" operation (applied to the datastore node) in the patch. When a new datastore node was created (other than an element in a list), a publisher will indicate a "create" operation in the patch. When a datastore node was deleted (other than an element in a list), the publisher indicates this by a "delete". When a new list element was created or removed, the publisher indicates it by an "insert" or "remove", respectively.

However a patch must be able to do more than just describe the delta from the previous state to the current state. As per Section 3.3, it must also be able to identify if transient changes have occurred on an object during a dampening period. To support this, it is valid to encode a YANG patch operation so that its application would result in no change between the previous and current state. This indicates that some churn has occurred on the object. An example of this would be a patch that does a "create" operation for a datastore node where the receiver believes one already exists, or a "merge" operation which replaces a previous value with the same value. Note that this means that the "create" and "delete" errors described in [RFC8072] section 2.5 are not errors, and are valid operations with YANG push.
3.6. Defining the Selection with a Datastore

A subscription must specify both the selection filters and the datastore against which these selection filters will be applied. This information is used to choose and subsequently push data from the publisher’s datastore to the receivers.

Only a single selection filter can be applied to a subscription at a time. An rpc request proposing a new selection filter MUST remove any existing filter. The following selection filter types are included in the yang-push data model, and may be applied against a datastore:

- subtree: A subtree selection filter identifies one or more datastore subtrees. When specified, update records will only come from the datastore nodes of selected datastore subtree(s). The syntax and semantics correspond to that specified for [RFC6241] section 6.

- xpath: An "xpath" selection filter is an XPath expression that returns a node set. When specified, updates will only come from the selected datastore nodes.

These filters are intended to be used as selectors that define which objects are within the scope of a subscription. A publisher MUST support at least one type of selection filter.

Xpath itself provides powerful filtering constructs and care must be used in filter definition. As an example, consider an XPath filter with a boolean result; such a result will not provide an easily interpretable subset of a datastore. Beyond the boolean example, it is quite possible to define an XPath filter where results are easy for an application to misinterpret. Consider an XPath filter which only passes a datastore object when an interface is up. It is up to the receiver to understand implications of the presence or absence of objects in each update.

When the set of selection filtering criteria is applied for a periodic subscription, all selected datastore nodes to which a receiver has access are provided to that receiver. If the same filtering criteria is applied to an on-change subscription, only the subset of those datastore nodes supporting on-change is provided. A datastore node which doesn’t support on-change is never sent as part of an on-change subscription’s "push-update" or "push-change-update".
3.7. Streaming Updates

Contrary to traditional data retrieval requests, datastore subscription enables an unbounded series of update records to be streamed over time. Two generic YANG notifications for update records have been defined for this: "push-update" and "push-change-update".

A "push-update" notification defines a complete, filtered update of the datastore per the terms of a subscription. This type of YANG notification is used for continuous updates of periodic subscriptions. A "push-update" notification can also be used for the on-change subscriptions in two cases. First it will be used as the initial "push-update" if there is a need to synchronize the receiver at the start of a new subscription. It also MAY be sent if the publisher later chooses to resynch an on-change subscription. The "push-update" update record contains an instantiated datastore subtree with all of the subscribed contents. The content of the update record is equivalent to the contents that would be obtained had the same data been explicitly retrieved using a datastore retrieval operation using the same transport with the same filters applied.

A "push-change-update" notification is the most common type of update for on-change subscriptions. The update record in this case contains the set of changes that datastore nodes have undergone since the last notification message. In other words, this indicates which datastore nodes have been created, deleted, or have had changes to their values. In cases where multiple changes have occurred and the object has not been deleted, the object’s most current value is reported. (In other words, for each object, only one change is reported, not its entire history. Doing so would defeat the purpose of the dampening period.)

These new "push-update" or "push-change-update" are encoded and placed within notification messages, and ultimately queued for egress over the specified transport.

The following is an example of a notification message for a subscription tracking the operational status of a single Ethernet port (per [RFC8343]). This notification message is encoded XML over NETCONF as per [I-D.draft-ietf-netconf-netconf-event-notifications].
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
<eventTime>2017-10-25T08:00:11.22Z</eventTime>
<subscription-id>1011</subscription-id>
<datastore-contents>
  <interfaces-state xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
    <interface>
      <name>eth0</name>
      <oper-status>up</oper-status>
    </interface>
  </interfaces-state>
</datastore-contents>
</push-update>
</notification>

Figure 1: Push example

The following is an example of an on-change notification message for the same subscription.

<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
<eventTime>2017-10-25T08:22:33.44Z</eventTime>
<subscription-id>89</subscription-id>
<datastore-changes>
  <yang-patch xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-patch">
    <patch-id>1</patch-id>
    <edit>
      <edit-id>edit1</edit-id>
      <operation>merge</operation>
      <target>/ietf-interfaces:interfaces-state</target>
      <value>
        <interfaces-state xmlns="http://foo.com/ietf-interfaces">
          <interface>
            <name>eth0</name>
            <oper-status>down</oper-status>
          </interface>
        </interfaces-state>
      </value>
    </edit>
  </yang-patch>
</datastore-changes>
</push-change-update>
</notification>

Figure 2: Push example for on change
Of note in the above example is the 'patch-id' with a value of '1'. Per [RFC8072], the 'patch-id' is an arbitrary string. With YANG Push, the publisher SHOULD put into the 'patch-id' a counter starting at '1' which increments with every 'push-change-update' generated for a subscription. If used as a counter, this counter MUST be reset to '1' anytime a resynchronization occurs (i.e., with the sending of a 'push-update'). Also if used as a counter, the counter MUST be reset to '1' the after passing a maximum value of '4294967295' (i.e. maximum value that can be represented using uint32 data type). Such a mechanism allows easy identification of lost or out-of-sequence update records.

3.8. Subscription Management

The RPCs defined within [I-D.draft-ietf-netconf-subscribed-notifications] have been enhanced to support datastore subscription negotiation. Also, new error codes have been added that are able to indicate why a datastore subscription attempt has failed, along with new yang-data that MAY be used to include details on input parameters that might result in a successful subsequent RPC invocation.

The establishment or modification of a datastore subscription can be rejected for multiple reasons. This includes a too large subtree request, or the inability of the publisher to push update records as frequently as requested. In such cases, no subscription is established. Instead, the subscription-result with the failure reason is returned as part of the RPC response. As part of this response, a set of alternative subscription parameters MAY be returned that would likely have resulted in acceptance of the subscription request. The subscriber may consider these as part of future subscription attempts.

In the case of a rejected request for an establishment of a datastore subscription, the hints MUST be transported within a yang-data "establish-subscription-datastore-error-info" container inserted into the RPC error response, in lieu of the "establish-subscription-stream-error-info" that is inserted in case of a stream subscription.

Below is a tree diagram for "establish-subscription-datastore-error-info". All tree diagrams used in this document follow the notation defined in [RFC8340]
Similarly, in the case of a rejected request for modification of a datastore subscription, the hints MUST be transported within a yang-data "modify-subscription-datastore-error-info" container inserted into the RPC error response, in lieu of the "modify-subscription-stream-error-info" that is inserted in case of a stream subscription.

Below is a tree diagram for "modify-subscription-datastore-error-info".

```
Figure 4: Tree diagram for modify-subscription-datastore-error-info
```

### 3.9. Receiver Authorization

A receiver of subscription data MUST only be sent updates for which they have proper authorization. A publisher MUST ensure that no non-authorized data is included in push updates. To do so, it needs to apply all corresponding checks applicable at the time of a specific pushed update and if necessary silently remove any non-authorized data from datastore subtrees. This enables YANG data pushed based on subscriptions to be authorized equivalently to a regular data retrieval (get) operation.

A publisher MUST allow for the possibility that a subscription’s selection filter references non-existent or access-protected data. Such support permits a receiver the ability to monitor the entire
lifecycle of some datastore tree. In this case, all "push-update" notifications must be sent empty, and no "push-change-update" notifications will be sent until some data becomes visible for a receiver.

A publisher MAY choose reject an establish-subscription request which selects non-existent or access-protected data. In addition, a publisher MAY choose to terminate a dynamic subscription or suspend a configured receiver when the authorization privileges of a receiver change, or the access controls for subscribed objects change. Such a capability enables the publisher to avoid having to support a continuous, and total filtering of an entire subscription’s content.

In these cases above, the error identity "unchanging-selection" SHOULD be returned. This reduces the possibility of leakage of access controlled objects.

Each "push-update" and "push-change-update" MUST have access control applied. This includes validating that read access is permitted for any new objects selected since the last notification message was sent to a particular each receiver. To accomplish this, implementations SHOULD support the conceptual authorization model of [RFC8342], specifically section 3.2.4.

<table>
<thead>
<tr>
<th>push-update or --&gt;</th>
<th>datastore node</th>
<th>yes</th>
<th>add datastore node</th>
</tr>
</thead>
<tbody>
<tr>
<td>push-change-update</td>
<td>access allowed?</td>
<td>---&gt;</td>
<td>to update message</td>
</tr>
</tbody>
</table>

Figure 5: Updated [rfc6536bis] access control for push updates

If read access into previously accessible nodes has been lost due to a receiver permissions change, this SHOULD be reported as a patch "delete" operation for on-change subscriptions. If not capable of handling such receiver permission changes with such a "delete", publisher implementations MUST force dynamic subscription re-establishment or configured subscription re-initialization so that appropriate filtering is installed.

3.10. On-change Notifiable YANG objects

In some cases, a publisher supporting on-change notifications may not be able to push updates for some object types on-change. Reasons for this might be that the value of the datastore node changes frequently (e.g., [RFC8343]’s in-octets counter), that small object changes are frequent and meaningless (e.g., a temperature gauge changing 0.1 degrees), or that the implementation is not capable of on-change notification for a particular object.
In those cases, it will be important for client applications to have a way to identify for which objects on-change notifications are supported and for which ones they are not supported. Otherwise client applications will have no way of knowing whether they can indeed rely on their on-change subscription to provide them with the change updates that they are interested in. In other words, if implementations do not provide a solution and do not support comprehensive on-change notifiability, clients of those implementations will have no way of knowing what their on-change subscription actually covers.

Implementations are therefore strongly advised to provide a solution to this problem. It is expected that such a solution will be standardized at some point in the future. In the meantime and until this occurs, implementations will be expected to provide their own solution.

3.11. Other Considerations

3.11.1. Robustness and reliability

Particularly in the case of on-change updates, it is important that these updates do not get lost. Or in case the loss of an update is unavoidable, it is critical that the receiver is notified accordingly.

Update records for a single subscription MUST NOT be resequenced prior to transport.

It is conceivable that under certain circumstances, a publisher will recognize that it is unable to include within an update record the full set of objects desired per the terms of a subscription. In this case, the publisher MUST take one or more of the following actions.

- A publisher MUST set the "incomplete-update" flag on any update record which is known to be missing information.
- It MAY choose to suspend a subscription as per [I-D.draft-ietf-netconf-subscribed-notifications].
- When resuming an on-change subscription, the publisher SHOULD generate a complete patch from the previous update record. If this is not possible and the "no-synch-on-start" option is not present for the subscription, then the full datastore contents MAY be sent via a "push-update" instead (effectively replacing the previous contents). If neither of these are possible, then an "incomplete-update" flag MUST be included on the next "push-change-update".
Note: It is perfectly acceptable to have a series of "push-change-update" notifications (and even "push update" notifications) serially queued at the transport layer awaiting transmission. It is not required to merge pending update messages. I.e., the dampening period applies to update record creation, not transmission.

3.11.2. Publisher capacity

It is far preferable to decline a subscription request than to accept such a request when it cannot be met.

Whether or not a subscription can be supported will be determined by a combination of several factors such as the subscription update trigger (on-change or periodic), the period in which to report changes (one second periods will consume more resources than one hour periods), the amount of data in the datastore subtree that is being subscribed to, and the number and combination of other subscriptions that are concurrently being serviced.

4. A YANG data model for management of datastore push subscriptions

4.1. Overview

The YANG data model for datastore push subscriptions is depicted in the following figure. The tree diagram follows the notation defined in [RFC8340]. New schema objects defined here (i.e., beyond those from [I-D.draft-ietf-netconf-subscribed-notifications]) are identified with "yp". For the reader’s convenience, in order to compact the tree representation, some nodes that are defined in ietf-subscribed-notifications and that are not essential to the understanding of the data model defined here have been removed. This is indicated by "..." in the diagram where applicable.

module: ietf-subscribed-notifications
...
  +--rw filters
  ...
  +--rw yp:selection-filter* [identifier]
     +--rw yp:identifier sn:filter-id
     +--rw (yp:filter-spec)?
      ++-(yp:datastore-subtree-filter)
      |  +--rw yp:datastore-subtree-filter? <anydata> {sn:subtree}?
      |  +--:(yp:datastore-xpath-filter)
      |     +--rw yp:datastore-xpath-filter? yang:xpath1.0 {sn:xpath}?
      +--rw subscriptions
       +--rw subscription* [identifier]
|  ...  
|  +--rw (target)  
|  |     +--:(stream)  
|  |     |     ...  
|  |     +--:(yp:datastore)  
|  |     |     +--rw yp:datastore identityref  
|  |     +--rw (yp:selection-filter)?  
|  |     |     +--:(yp:by-reference)  
|  |     |     |     +--rw yp:selection-filter-ref  
|  |     |     |     selection-filter-ref  
|  |     |     +--:(yp:within-subscription)  
|  |     |     +--rw (yp:filter-spec)?  
|  |     |     |     +--:(yp:datastore-subtree-filter)  
|  |     |     |     |     +--rw yp:datastore-subtree-filter?  
|  |     |     |     |     |     <anydata> {sn:subtree}?  
|  |     |     |     +--:(yp:datastore-xpath-filter)  
|  |     |     |     |     +--rw yp:datastore-xpath-filter?  
|  |     |     |     |     |     yang:xpath1.0 {sn:xpath}?  
|  |     |     ...  
|  |     +--rw (yp:update-trigger)?  
|  |     +--:(yp:periodic)  
|  |     |     +--rw yp:periodic!  
|  |     |     |     +--rw yp:period yang:timeticks  
|  |     |     |     +--rw yp:anchor-time? yang:date-and-time  
|  |     |     +--:(yp:on-change) {on-change}?  
|  |     |     +--rw yp:on-change!  
|  |     |     |     +--rw yp:dampening-period? yang:timeticks  
|  |     |     |     +--rw yp:no-synch-on-start? empty  
|  |     |     +--rw yp:excluded-change* change-type  

rpcs:  
|  +---x establish-subscription  
|  |     +---w input  
|  ...  
|  +--w (target)  
|  |     +--:(stream)  
|  |     |     ...  
|  |     +--:(yp:datastore)  
|  |     |     +--w yp:datastore identityref  
|  |     +--w (yp:selection-filter)?  
|  |     |     +--:(yp:by-reference)  
|  |     |     |     +--w yp:selection-filter-ref  
|  |     |     |     selection-filter-ref  
|  |     |     +--:(yp:within-subscription)  
|  |     |     +--w (yp:filter-spec)?  
|  |     |     |     +--:(yp:datastore-subtree-filter)  
|  |     |     |     |     +--w yp:datastore-subtree-filter?  
|  |     |     |     |     |     <anydata> {sn:subtree}?  

yang-data (for placement into rpc error responses)
...

notifications:
+---n replay-completed (replay)?
| ...
+---n subscription-completed
| ...
+---n subscription-started (configured)?
| ...
+--ro (target)
| ...
+--: (yp:datastore)
  +--ro yp:datastore                   identityref
  +---ro (yp:selection-filter)?
     +--:(yp:by-reference)
      |  +--ro yp:selection-filter-ref
      |     selection-filter-ref
     +--:(yp:within-subscription)
      +--ro (yp:filter-spec)?
         +--:(yp:datastore-subtree-filter)
          |  +--ro yp:datastore-subtree-filter?
             <anydata> {sn:subtree}?
          +--ro yp:datastore-xpath-filter?
             yang:xpath1.0 (sn:xpath)?
| ...
+--ro (yp:update-trigger)?
     +--:(yp:periodic)
      |  +--ro yp:period                  yang:timeticks
      |  +--ro yp:anchor-time?            yang:date-and-time
     +--:(yp:on-change) (on-change)?
      +--ro yp:on-change!
       +--ro yp:dampening-period?       yang:timeticks
       +--ro yp:no-synch-on-start?       empty
       +--ro yp:excluded-change*         change-type
+---n subscription-resumed
| ...
+---n subscription-modified
| ...
+--ro (target)
| ...
+--: (yp:datastore)
  +--ro yp:datastore                   identityref
  +---ro (yp:selection-filter)?
     +--:(yp:by-reference)
      |  +--ro yp:selection-filter-ref

module: ietf-yang-push

rpcs:
  +---x resynch-subscription {on-change}?
  +---w input
    +---w identifier    sn:subscription-id

yang-data: (for placement into rpc error responses)
  +--- resynch-subscription-error
    +---ro reason?                   identityref
    +---ro period-hint?              timeticks
    +---ro filter-failure-hint?      string
    +---ro object-count-estimate?    uint32
    +---ro object-count-limit?       uint32
    +---ro kilobytes-estimate?       uint32
    +---ro kilobytes-limit?          uint32
  +--- establish-subscription-error-datastore
    +---ro reason?                   identityref
    +---ro period-hint?              timeticks
    +---ro filter-failure-hint?      string
    +---ro object-count-estimate?    uint32
    +---ro object-count-limit?       uint32
    +---ro kilobytes-estimate?       uint32
    +---ro kilobytes-limit?          uint32
notifications:
  +++n push-update
  | +++ro subscription-id? sn:subscription-id
  | +++ro incomplete-update? empty
  | +++ro datastore-contents? <anydata>
  +++n push-change-update {on-change}?
  | +++ro subscription-id? sn:subscription-id
  | +++ro incomplete-update? empty
  | +++ro datastore-changes? <anydata>

Figure 6: Model structure

Selected components of the model are summarized below.

4.2. Subscription configuration

Both configured and dynamic subscriptions are represented within the list "subscription". New parameters extending the basic subscription data model in [I-D.draft-ietf-netconf-subscribed-notifications] include:

- The targeted datastore from which the selection is being made. The potential datastores include those from [RFC8341]. A platform may also choose to support a custom datastore.

- A selection filter identifying yang nodes of interest within a datastore. Filter contents are specified via a reference to an existing filter, or via an in-line definition for only that subscription. Referenced filters allows an implementation to avoid evaluating filter acceptability during a dynamic subscription request. The case statement differentiates the options.

- For periodic subscriptions, triggered updates will occur at the boundaries of a specified time interval. These boundaries can be calculated from the periodic parameters:
* a "period" which defines the duration between push updates.

* an "anchor-time"; update intervals always fall on the points in time that are a multiple of a "period" from an "anchor-time". If "anchor-time" is not provided, then the "anchor-time" MUST be set with the creation time of the initial update record.

For on-change subscriptions, assuming any dampening period has completed, triggering occurs whenever a change in the subscribed information is detected. On-change subscriptions have more complex semantics that is guided by its own set of parameters:

* a "dampening-period" specifies the interval that must pass before a successive update for the subscription is sent. If no dampening period is in effect, the update is sent immediately. If a subsequent change is detected, another update is only sent once the dampening period has passed for this subscription.

* an "excluded-change" flag which allows restriction of the types of changes for which updates should be sent (e.g., only add to an update record on object creation).

* a "no-synch-on-start" flag which specifies whether a complete update with all the subscribed data is to be sent at the beginning of a subscription.

4.3. YANG Notifications

4.3.1. State Change Notifications

Subscription state notifications and mechanism are reused from [I-D.draft-ietf-netconf-subscribed-notifications]. Notifications "subscription-started" and "subscription-modified" have been augmented to include the datastore specific objects.

4.3.2. Notifications for Subscribed Content

Along with the subscribed content, there are other objects which might be part of a "push-update" or "push-change-update" notification.

A "subscription-id" MUST be transported along with the subscribed contents. An [RFC5277] Section 4 one-way notification MAY be used for encoding updates. Where it is, the relevant "subscription-id" MUST be encoded as the first element within each "push-update" or "push-change-update". This allows a receiver to differentiate which subscription resulted in a particular push.
A "time-of-update" which represents the time an update record snapshot was generated. A receiver MAY assume that a publisher’s objects have these pushed values at this point in time.

An "incomplete-update" leaf. This leaf indicates that not all changes which have occurred since the last update are actually included with this update. In other words, the publisher has failed to fulfill its full subscription obligations. (For example a datastore was unable to providing the full set of datastore nodes to a publisher process.) To facilitate re-synchronization of on-change subscriptions, a publisher MAY subsequently send a "push-update" containing a full selection snapshot of subscribed data.

4.4. YANG RPCs

YANG-Push subscriptions are established, modified, and deleted using RPCs augmented from [I-D.draft-ietf-netconf-subscribed-notifications].

4.4.1. Establish-subscription RPC

The subscriber sends an establish-subscription RPC with the parameters in section 3.1. An example might look like:

```xml
<netconf:rpc message-id="101"
xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
<establish-subscription
xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
<yp:datastore>
ds:operational
</yp:source>
<xpath-filter
xmlns:ex="http://example.com/sample-data/1.0"
select="/ex:foo"/>
</yp:datastore>
<yp:periodic>
<yp:period>500</yp:period>
</yp:periodic>
</establish-subscription>
</netconf:rpc>
```

Figure 7: Establish-subscription RPC

A positive response includes the "identifier" of the accepted subscription. In that case a publisher MAY respond:
A subscription can be rejected for multiple reasons, including the lack of authorization to establish a subscription, no capacity to serve the subscription at the publisher, or the inability of the publisher to select datastore content at the requested cadence.

If a request is rejected because the publisher is not able to serve it, the publisher SHOULD include in the returned error hints which help a subscriber understand subscription parameters might have been accepted for the request. These hints would be included within the yang-data structure "establish-subscription-error-datastore". However even with these hints, there are no guarantee that subsequent requests will in fact be accepted.

The specific parameters to be returned in as part of the RPC error response depend on the specific transport that is used to manage the subscription. In the case of NETCONF [I-D.draft-ietf-netconf-netconf-event-notifications], when a subscription request is rejected, the NETCONF RPC reply MUST include an "rpc-error" element with the following elements:

- "error-type" of "application".
- "error-tag" of "operation-failed".
- Optionally, an "error-severity" of "error" (this MAY but does not have to be included).
- "error-app-tag" with the value being a string that corresponds to an identity with a base of "establish-subscription-error".
- Optionally, "error-info" containing XML-encoded data with hints for parameter settings that might result in future RPC success per yang-data definition "establish-subscription-error-datastore".
For example, for the following request:

```xml
<netconf:rpc message-id="101"
    xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
    <establish-subscription
        xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
        <yp:datastore
            ds:operational
        </yp:datastore>
        <yp:datastore-xpath-filter netconf:type="xpath"
            xmlns:ex="http://example.com/sample-data/1.0">
            /ex:foo
        </yp:datastore-xpath-filter>
        <yp:on-change>
            <yp:dampening-period>100</yp:dampening-period>
        </yp:on-change>
    </establish-subscription>
</netconf:rpc>
```

Figure 9: Establish-subscription request example 2

a publisher that cannot serve on-change updates but periodic updates might return the following:

```xml
<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <rpc-error
        <error-type>application</error-type>
        <error-tag>operation-failed</error-tag>
        <error-severity>error</error-severity>
        <error-app-tag>
            on-change-unsupported
        </error-app-tag>
        <error-message
        </error-message>
        <error-path
            /yp:periodic/yp:period
        </error-path>
    </rpc-error>
</rpc-reply>
```

Figure 10: Establish-subscription error response example 2
4.4.2. Modify-subscription RPC

The subscriber MAY invoke the "modify-subscription" RPC for a subscription it previously established. The subscriber will include newly desired values in the "modify-subscription" RPC. Parameters not included MUST remain unmodified. Below is an example where a subscriber attempts to modify the "period" of a subscription.

```
<netconf:rpc message-id="102"
 xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <modify-subscription
      xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
    <identifier>1011</identifier>
    <yp:datastore
      ds:operational
    </yp:datastore>
    <yp:datastore-xpath-filter
       netconf:type="xpath" xmlns:ex="http://example.com/sample-data/1.0">
      /ex:bar
    </yp:datastore-xpath-filter>
    <yp:periodic>
      <yp:period>250</yp:period>
    </yp:periodic>
  </modify-subscription>
</netconf:rpc>
```

Figure 11: Modify subscription request

The publisher MUST respond explicitly positively or negatively to the request. If the subscription modification is rejected, the subscription is maintained as it was before the modification request. In addition, the publisher MUST send an rpc error response. This rpc error response may contain hints encapsulated within the yang-data structure "modify-subscription-error-datastore". A subscription MAY be modified multiple times.

The specific parameters to be returned in as part of the RPC error response depend on the specific transport that is used to manage the subscription. In the case of NETCONF [I-D.draft-ietf-netconf-netconf-event-notifications], when a subscription request is rejected, the NETCONF RPC reply MUST include an "rpc-error" element with the following elements:

- "error-type" of "application".
- "error-tag" of "operation-failed".
o Optionally, an "error-severity" of "error" (this MAY but does not have to be included).

o "error-app-tag" with the value being a string that corresponds to an identity with a base of "modify-subscription-error".

o "error-path" pointing to the object or parameter that caused the rejection.

o Optionally, "error-info" containing XML-encoded data with hints for parameter settings that might result in future RPC success per yang-data definition "modify-subscription-error-datastore".

A configured subscription cannot be modified using "modify-subscription" RPC. Instead, the configuration needs to be edited as needed.

4.4.3. Delete-subscription RPC

To stop receiving updates from a subscription and effectively delete a subscription that had previously been established using an "establish-subscription" RPC, a subscriber can send a "delete-subscription" RPC, which takes as only input the subscription’s "identifier". This RPC is unmodified from [I-D.draft-ietf-netconf-subscribed-notifications].

4.4.4. Resynch-subscription RPC

This RPC is only applicable only for on-change subscriptions previously established using an "establish-subscription" RPC. For example:

```xml
<netconf:rpc message-id="103"
xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <resynch-subscription
xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push"
    <identifier>1011</identifier>
  </resynch-subscription>
</netconf:rpc>
```

Resynch subscription

On receipt, a publisher must either accept the request and quickly follow with a "push-update", or send an appropriate error within an rpc error response. Within an error response, the publisher may include supplemental information about the reasons within the yang-data structure "resynch-subscription-error".
4.4.5. YANG Module Synchronization

To make subscription requests, the subscriber needs to know the YANG module library available on the publisher. The YANG 1.0 module library information is sent by a NETCONF server in the NETCONF "hello" message. For YANG 1.1 modules and all modules used with the RESTCONF [RFC8040] protocol, this information is provided by the YANG Library module (ietf-yang-library.yang from [RFC7895]). This YANG library information is important for the receiver to reproduce the set of object definitions used within the publisher.

The YANG library includes a module list with the name, revision, enabled features, and applied deviations for each YANG module implemented by the publisher. The receiver is expected to know the YANG library information before starting a subscription. The "/modules-state/module-set-id" leaf in the "ietf-yang-library" module can be used to cache the YANG library information.

The set of modules, revisions, features, and deviations can change at run-time (if supported by the publisher implementation). In this case, the receiver needs to be informed of module changes before datastore nodes from changed modules can be processed correctly. The YANG library provides a simple "yang-library-change" notification that informs the subscriber that the library has changed. The receiver then needs to re-read the entire YANG library data for the replicated publisher in order to detect the specific YANG library changes. The "ietf-netconf-notifications" module defined in [RFC6470] contains a "netconf-capability-change" notification that can identify specific module changes. For example, the module URI capability of a newly loaded module will be listed in the "added-capability" leaf-list, and the module URI capability of an removed module will be listed in the "deleted-capability" leaf-list.

5. YANG module

<CODE BEGINS> file "ietf-yang-push@2018-07-01.yang"
module ietf-yang-push {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-push";
  prefix yp;

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }
  import ietf-subscribed-notifications {
    prefix sn;

import ietf-datastores {
    prefix ds;
    reference
    "RFC 8342: Network Management Datastore Architecture (NMDA)";
}
import ietf-restconf {
    prefix rc;
    reference
    "RFC 8040: RESTCONF Protocol";
}

organization "IETF";
contact
"WG Web:  <http://tools.ietf.org/wg/netconf/>
WG List:  <mailto:netconf@ietf.org>
Editor:  Alexander Clemm
         <mailto:ludwig@clemm.org>
Editor:  Eric Voit
         <mailto:evoit@cisco.com>
Editor:  Alberto Gonzalez Prieto
         <mailto:agonzalezpri@vmware.com>
Editor:  Ambika Prasad Tripathy
         <mailto:ambtripa@cisco.com>
Editor:  Einar Nilsen-Nygaard
         <mailto:einarann@cisco.com>
Editor:  Andy Bierman
         <mailto:andy@yumaworks.com>
Editor:  Balazs Lengyel
         <mailto:balazs.lengyel@ericsson.com>";

description
"This module contains YANG specifications for YANG push."
revision 2018-07-01 {
    description
        "Initial revision."
        NOTE TO RFC EDITOR:
            (1) Please replace the above revision date to the date of RFC publication when published.
            (2) Please replace the date in the file name (ietf-yang-push@2018-07-01.yang) to the date of RFC publication.
            (3) Please replace the following reference to draft-ietf-netconf-yang-push-17 with RFC number when published (i.e. RFC xxxx)."
    reference
        "draft-ietf-netconf-yang-push-17";
}

feature on-change {
    description
        "This feature indicates that on-change triggered subscriptions are supported."
}

/*
 * IDENTITIES
 */

/* Error type identities for datastore subscription */
identity resynch-subscription-error {
    description
    "Problem found while attempting to fulfill an 'resynch-subscription' RPC request. ";
}

identity cant-exclude {
    base sn:establish-subscription-error;
    description
    "Unable to remove the set of 'excluded-changes'. This means the publisher is unable to restrict 'push-change-update's to just the change types requested for this subscription.";
}

identity datastore-not-subscribable {
    base sn:establish-subscription-error;
    base sn:subscription-terminated-reason;
    description
    "This is not a subscribable datastore.";
}

identity no-such-subscription-resynch {
    base resynch-subscription-error;
    description
    "Referenced subscription doesn't exist. This may be as a result of a non-existent subscription ID, an ID which belongs to another subscriber, or an ID for configured subscription.";
}

identity on-change-unsupported {
    base sn:establish-subscription-error;
    description
    "On-change is not supported for any objects which are selectable by this filter.";
}

identity on-change-synch-unsupported {
    base sn:establish-subscription-error;
    description
    "Neither synch on start nor resynchronization are supported for this subscription. This error will be used for two reasons. First if an 'establish-subscription' RPC doesn't include 'no-synch-on-start', yet the publisher can't support sending a 'push-update' for this subscription for reasons other than 'on-change-unsupported' or 'synchronization-size'. And second, if the 'resynch-subscription' RPC is invoked either for an existing periodic subscription, or for an on-change subscription which can't support resynchronization.";
identity period-unsupported {
  base sn:establish-subscription-error;
  base sn:modify-subscription-error;
  base sn:subscription-suspended-reason;
  description
      "Requested time period is too short. This can be for both periodic and on-change subscriptions (with or without dampening.) Hints suggesting alternative periods may be returned as supplemental information.";
}

identity result-too-big {
  base sn:establish-subscription-error;
  base sn:modify-subscription-error;
  base sn:subscription-suspended-reason;
  description
      "Periodic or on-change push update datatrees exceed a maximum size limit. Hints on estimated size of what was too big may be returned as supplemental information.";
}

identity synchronization-size {
  base sn:establish-subscription-error;
  base sn:modify-subscription-error;
  base resynch-subscription-error;
  base sn:subscription-suspended-reason;
  description
      "Synch-on-start or resynchronization datatree exceeds a maximum size limit. Hints on estimated size of what was too big may be returned as supplemental information.";
}

identity unchanging-selection {
  base sn:establish-subscription-error;
  base sn:modify-subscription-error;
  base sn:subscription-terminated-reason;
  description
      "Selection filter is unlikely to ever select datatree nodes. This means that based on the subscriber's current access rights, the publisher recognizes that the selection filter is unlikely to ever select datatree nodes which change. Examples for this might be that node or subtree doesn't exist, read access is not permitted for a receiver, or static objects that only change at reboot have been chosen.";
}
typedef change-type {
    type enumeration {
        enum "create" {
            description "A change that refers to the creation of a new data node.";
        }
        enum "delete" {
            description "A change that refers to the deletion of a data node.";
        }
        enum "insert" {
            description "A change that refers to the insertion of a new user-ordered data node.";
        }
        enum "merge" {
            description "A change that refers to a merging of a new value with a target data node.";
        }
        enum "move" {
            description "A change that refers to a reordering of the target data node";
        }
        enum "replace" {
            description "A change that refers to a replacement of the target data node’s value.";
        }
        enum "remove" {
            description "A change that refers to the removal of a data node.";
        }
    }
    description "Specifies different types of datastore changes.";
    reference "RFC 8072 section 2.5, with a delta that it is valid for a receiver to process an update record which performs a create operation on a datastore node the receiver believes exists, or to process a delete on a datastore node the receiver believes is missing.";
}
typedef selection-filter-ref {
    type leafref {
        path "/sn:filters/yp:selection-filter/yp:filter-id";
        description "This type is used to reference a selection filter."
    }
}

/*
* GROUP DEFINITIONS
*/

grouping datastore-criteria {
    description "A grouping to define criteria for which selected objects from
    a targeted datastore should be included in push updates.";
    leaf datastore {
        type identityref {
            base ds:datastore;
        }
        mandatory true;
        description "Datastore from which to retrieve data.";
    }
    uses selection-filter-objects;
}

grouping selection-filter-types {
    description "This grouping defines the types of selectors for objects from a
datastore.";
    choice filter-spec {
        description "The content filter specification for this request.";
        anydata datastore-subtree-filter {
            if-feature "sn:subtree";
            description "This parameter identifies the portions of the
target datastore to retrieve.";
        }
        leaf datastore-xpath-filter {
            if-feature "sn:xpath";
            type yang:xpath1.0;
            description "This parameter contains an XPath expression identifying the
portions of the target datastore to retrieve.

If the expression returns a node-set, all nodes in the
node-set are selected by the filter. Otherwise, if the expression does not return a node-set, the filter doesn’t select any nodes.

The expression is evaluated in the following XPath context:

- The set of namespace declarations are those in scope on the ‘datastore-xpath-filter’ leaf element.
- The set of variable bindings is empty.
- The function library is the core function library, and the XPath functions defined in section 10 in RFC 7950.
- The context node is the root node of the target datastore.
grouping update-policy-modifiable {
  description
    "This grouping describes the datastore specific subscription
    conditions that can be changed during the lifetime of the
    subscription.";
  choice update-trigger {
    description
      "Defines necessary conditions for sending an event record to
      the subscriber.";
    case periodic {
      container periodic {
        presence "indicates an periodic subscription";
        description
          "The publisher is requested to notify periodically the
           current values of the datastore as defined by the selection
           filter.";
        leaf period {
          type yang:timeticks;
          mandatory true;
          description
            "Duration of time which should occur between periodic
             push updates.";
        }
        leaf anchor-time {
          type yang:date-and-time;
          description
            "Designates a timestamp before or after which a series of
             periodic push updates are determined. The next update
             will take place at a whole multiple interval from the
             anchor time. For example, for an anchor time is set for
             the top of a particular minute and a period interval of a
             minute, updates will be sent at the top of every minute
             this subscription is active.";
        }
      }
    case on-change {
      if-feature "on-change";
      container on-change {
        presence "indicates an on-change subscription";
        description
          "The publisher is requested to notify changes in values in
           the datastore subset as defined by a selection filter.";
        leaf dampening-period {
          type yang:timeticks;
          default 0;
        }
      }
    }
  }
}
description
"Specifies the minimum interval between the assembly of successive update records for a single receiver of a subscription. Whenever subscribed objects change, and a dampening period interval (which may be zero) has elapsed since the previous update record creation for a receiver, then any subscribed objects and properties which have changed since the previous update record will have their current values marshalled and placed into a new update record."
}
}

grouping update-policy {

description
"This grouping describes the datastore specific subscription conditions of a subscription."
;
uses update-policy-modifiable {
	augment "update-trigger/on-change/on-change" {

description
"Includes objects not modifiable once subscription is established.";

leaf no-synch-on-start {

type empty;

description
"The presence of this object restricts an on-change subscription from sending push-update notifications. When present, pushing a full selection per the terms of the selection filter MUST NOT be done for this subscription. Only updates about changes, i.e. only push-change-update notifications are sent. When absent (default behavior), in order to facilitate a receiver’s synchronization, a full update is sent when the subscription starts using a push-update notification. After that, push-change-update notifications are exclusively sent unless the publisher chooses to resynch the subscription via a new push-update notification."
;

leaf-list excluded-change {

type change-type;

description
"Use to restrict which changes trigger an update. For example, if modify is excluded, only creation and deletion of objects is reported.";
}

grouping hints {
  description
    "Parameters associated with some error for a subscription made
    upon a datastore."
  leaf period-hint {
    type yang:timeticks;
    description
      "Returned when the requested time period is too short. This
      hint can assert a viable period for either a periodic push
      cadence or an on-change dampening interval."
  }
  leaf filter-failure-hint {
    type string;
    description
      "Information describing where and/or why a provided filter
      was unsupportable for a subscription."
  }
  leaf object-count-estimate {
    type uint32;
    description
      "If there are too many objects which could potentially be
      returned by the selection filter, this identifies the estimate
      of the number of objects which the filter would potentially
      pass."
  }
  leaf object-count-limit {
    type uint32;
    description
      "If there are too many objects which could be returned by the
      selection filter, this identifies the upper limit of the
      publisher’s ability to service for this subscription."
  }
  leaf kilobytes-estimate {
    type uint32;
    description
      "If the returned information could be beyond the capacity of
      the publisher, this would identify the data size which could
      result from this selection filter."
  }
  leaf kilobytes-limit {
    type uint32;
    description
      "If the returned information would be beyond the capacity of
      the publisher, this identifies the upper limit of the
This RPC allows a subscriber of an active on-change subscription to request a full push of objects. A successful invocation results in a push-update of all datastore objects that the subscriber is permitted to access. This RPC can only be invoked on the same session on which the subscription was established (using an establish-subscription RPC). In case of an error, a resynch-subscription-error is sent as part of an error response.

input {
  leaf identifier {
    type sn:subscription-id;
    mandatory true;
    description "Identifier of the subscription that is to be resynched.";
  }
}

c:yang-data resynch-subscription-error {
  container resynch-subscription-error {
    description "If a 'resynch-subscription' RPC fails, the subscription is not resynched and the RPC error response MUST indicate the reason for this failure. This yang-data MAY be inserted as structured data within a subscription's RPC error response to indicate the failure reason."
    leaf reason {
      type identityref {
        base resynch-subscription-error;
      } mandatory true;
      description "Indicates the reason why the publisher has declined a request for subscription resynchronization."
    }
    uses hints;
  }
}
augment "/sn:establish-subscription/sn:input" {
  description
  "This augmentation adds additional subscription parameters that
  apply specifically to datastore updates to RPC input.";
  uses update-policy;
}

augment "/sn:establish-subscription/sn:input/sn:target" {
  description
  "This augmentation adds the datastore as a valid target
  for the subscription to RPC input.";
  case datastore {
    description
    "Information specifying the parameters of an request for a
    datastore subscription.";
    uses datastore-criteria;
  }
}

rc:yang-data establish-subscription-datastore-error-info {
  container establish-subscription-datastore-error-info {
    description
    "If any 'establish-subscription' RPC parameters are
    unsupportable against the datastore, a subscription is not
    created and the RPC error response MUST indicate the reason why
    the subscription failed to be created. This yang-data MAY be
    inserted as structured data within a subscription's RPC error
    response to indicate the failure reason. This yang-data MUST be
    inserted if hints are to be provided back to the subscriber.";
    leaf reason {
      type identityref {
        base sn:establish-subscription-error;
      }
      description
      "Indicates the reason why the subscription has failed to
      be created to a targeted datastore.";
      uses hints;
    }
  }
}

augment "/sn:modify-subscription/sn:input" {
  description
  "This augmentation adds additional subscription parameters
  specific to datastore updates.";

uses update-policy-modifiable;
}

augment "/sn:modify-subscription/sn:input/sn:target" {
  description
  "This augmentation adds the datastore as a valid target
  for the subscription to RPC input.";
  case datastore {
    description
    "Information specifying the parameters of an request for a
datastore subscription.";
    uses selection-filter-objects;
  }
}

rc:yang-data modify-subscription-datastore-error-info {
  container modify-subscription-datastore-error-info {
    description
    "This yang-data MAY be provided as part of a subscription’s RPC
error response when there is a failure of a
‘modify-subscription’ RPC which has been made against a
datastore. This yang-data MUST be used if hints are to be
provides back to the subscriber.";
    leaf reason {
      type identityref {
        base sn:modify-subscription-error;
      }
      description
      "Indicates the reason why the subscription has failed to
be modified.";
    }
    uses hints;
  }
}

/*
 * NOTIFICATIONS
 */

notification push-update {
  description
  "This notification contains a push update, containing data
subscribed to via a subscription. This notification is sent for
periodic updates, for a periodic subscription. It can also be
used for synchronization updates of an on-change subscription.
This notification shall only be sent to receivers of a
subscription. It does not constitute a general-purpose
notification that would be subscribable as part of the NETCONF
event stream by any receiver.

leaf subscription-id {
    type sn:subscription-id;
    description
    "This references the subscription which drove the notification
to be sent."
}

leaf incomplete-update {
    type empty;
    description
    "This is a flag which indicates that not all datastore nodes
subscribed to are included with this update. In other words,
the publisher has failed to fulfill its full subscription
obligations, and despite its best efforts is providing an
incomplete set of objects."
}

anydata datastore-contents {
    description
    "This contains the updated data. It constitutes a snapshot
at the time-of-update of the set of data that has been
subscribed to. The snapshot corresponds to the same
snapshot that would be returned in a corresponding get
operation with the same selection filter parameters
applied."
}

notification push-change-update {
    if-feature "on-change";
    description
    "This notification contains an on-change push update. This
notification shall only be sent to the receivers of a
subscription; it does not constitute a general-purpose
notification."

leaf subscription-id {
    type sn:subscription-id;
    description
    "This references the subscription which drove the notification
to be sent."
}

leaf incomplete-update {
    type empty;
    description
    "The presence of this object indicates not all changes which
have occurred since the last update are included with this
update. In other words, the publisher has failed to
fulfill its full subscription obligations, for example in
cases where it was not able to keep up with a change burst.";
anydata datastore-changes {
  description
    "This contains the set of datastore changes of the
    target datastore starting at the time of the
    previous update, per the terms of the subscription.
    The datastore changes are encoded per RFC 8027
    (YANG Patch).";
}
}

augment "/sn:subscription-started" {
  description
    "This augmentation adds datastore-specific objects to
    the notification that a subscription has started.";
  uses update-policy;
}

augment "/sn:subscription-started/sn:target" {
  description
    "This augmentation allows the datastore to be included as part
    of the notification that a subscription has started.";
  case datastore {
    uses datastore-criteria {
      refine "selection-filter/within-subscription" {
        description
          "Specifies the selection filter and where it originated
          from. If the 'selection-filter-ref' is populated,
          the filter within the subscription came from the 'filters'
          container. Otherwise it is populated in-line as part of the
          subscription itself.";
      }
    }
  }
}

augment "/sn:subscription-modified" {
  description
    "This augmentation adds datastore-specific objects to
    the notification that a subscription has been modified.";
  uses update-policy;
}

augment "/sn:subscription-modified/sn:target" {
  description
    "This augmentation allows the datastore to be included as part
    of the notification that a subscription has been modified.";
  case datastore {

uses datastore-criteria {
    refine "selection-filter/within-subscription" {
        description
            "Specifies where the selection filter, and where it came from within the subscription and then populated within this notification. If the 'selection-filter-ref' is populated, the filter within the subscription came from the 'filters' container. Otherwise it is populated in-line as part of the subscription itself."
    }
}

augment "/sn:filters" {
    description
        "This augmentation allows the datastore to be included as part of the selection filtering criteria for a subscription."
    list selection-filter {
        key "filter-id";
        description
            "A list of pre-configured filters that can be applied to datastore subscriptions."
        leaf filter-id {
            type string;
            description
                "An identifier to differentiate between selection filters."
        }
        uses selection-filter-types;
    }
}

augment "/sn:subscriptions/sn:subscription" {
    description
        "This augmentation adds many datastore specific objects to a subscription."
    uses update-policy;
}

augment "/sn:subscriptions/sn:subscription/sn:target" {
    description
        "This augmentation allows the datastore to be included as part of the selection filtering criteria for a subscription."
    case datastore {
        uses datastore-criteria;
    }
}
6. IANA Considerations

   This document registers the following namespace URI in the "IETF XML Registry" [RFC3688]:

   Registrant Contact: The IESG.
   XML: N/A; the requested URI is an XML namespace.

   This document registers the following YANG module in the "YANG Module Names" registry [RFC6020]:

   Name: ietf-yang-push
   Prefix: yp
   Reference: draft-ietf-netconf-yang-push-17.txt (RFC form)

7. Security Considerations

   The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

   The NETCONF access control model [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

   There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:
First, it should be noted that the YANG module augments the YANG module from [I-D.draft-ietf-netconf-subscribed-notifications]. All security considerations that are listed there are relevant also for datastore subscriptions. In the following, we focus on the data nodes that are newly introduced here.

Subtree "selection-filter" under container "filters": This subtree allows to specify which objects or subtrees to include in a datastore subscription. An attacker could attempt to modify the filter. For example, the filter might be modified to result in very few objects being filtered in order to attempt to overwhelm the receiver. Alternatively, the filter might be modified to result in certain objects to be excluded from updates, in order to have certain changes go unnoticed.

Subtree "datastore" in choice "target" in list "subscription": Analogous to "selection filter", an attacker might attempt to modify the objects being filtered in order to overwhelm a receiver with a larger volume of object updates than expected, or to have certain changes go unnoticed.

Choice "update-trigger" in list "subscription": By modifying the update trigger, an attacker might alter the updates that are being sent in order to confuse a receiver, to withhold certain updates to be sent to the receiver, and/or to overwhelm a receiver. For example, an attacker might modify the period with which updates are reported for a periodic subscription, or it might modify the dampening period for an on-change subscription, resulting in greater delay of successive updates (potentially affecting responsiveness of applications that depend on the updates) or in a high volume of updates (to exhaust receiver resources).

RPC "resynch-subscription": This RPC allows a subscriber of an on-change subscription to request a full push of objects in the subscription’s scope. This can result in a large volume of data. An attacker could attempt to use this RPC to exhaust resources on the server to generate the data, and attempt to overwhelm a receiver with the resulting data volume.

8. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Tim Jenkins, Martin Bjorklund, Kent Watsen, Susan Hares, Yang Geng, Peipei Guo, Michael Scharf, Guangying Zheng, and Tom Fetch.
9. References

9.1. Normative References

[I-D.draft-ietf-netconf-subscribed-notifications]


9.2. Informative References


Appendix A. Appendix A: Subscription Errors

A.1. RPC Failures

Rejection of an RPC for any reason is indicated by via RPC error response from the publisher. Valid RPC errors returned include both existing transport layer RPC error codes, such as those seen with NETCONF in [RFC6241], as well as subscription specific errors such as those defined within the YANG model. As a result, how subscription errors are encoded within an RPC error response is transport dependent.

References to specific identities within the either the subscribed-notifications YANG model or the yang-push YANG model may be returned as part of the error responses resulting from failed attempts at datastore subscription. Following are valid errors per RPC (note: throughout this section the prefix ‘sn’ indicates an item imported from the subscribed-notifications.yang model):

<table>
<thead>
<tr>
<th>establish-subscription</th>
<th>modify-subscription</th>
</tr>
</thead>
<tbody>
<tr>
<td>cant-exclude</td>
<td>sn:filter-unsupported</td>
</tr>
<tr>
<td>datastore-not-subscribable</td>
<td>sn:insufficient-resources</td>
</tr>
<tr>
<td>sn:dscp-unavailable</td>
<td>sn:no-such-subscription</td>
</tr>
<tr>
<td>sn:filter-unsupported</td>
<td>period-unsupported</td>
</tr>
<tr>
<td>sn:insufficient-resources</td>
<td>result-too-big</td>
</tr>
<tr>
<td>on-change-unsupported</td>
<td>synchronization-size</td>
</tr>
<tr>
<td>on-change-synch-unsupported</td>
<td>unchanging-selection</td>
</tr>
<tr>
<td>period-unsupported</td>
<td>result-too-big</td>
</tr>
<tr>
<td>result-too-big</td>
<td>synchronization-size</td>
</tr>
<tr>
<td>synchronization-size</td>
<td>unchanging-selection</td>
</tr>
<tr>
<td>unchanging-selection</td>
<td>resynch-subscription</td>
</tr>
<tr>
<td>delete-subscription</td>
<td>kill-subscription</td>
</tr>
<tr>
<td>sn:no-such-subscription</td>
<td>no-such-subscription-resynch</td>
</tr>
<tr>
<td></td>
<td>synchronization-size</td>
</tr>
</tbody>
</table>

There is one final set of transport independent RPC error elements included in the YANG model. These are the following four yang-data structures for failed datastore subscriptions:
1. yang-data establish-subscription-error-datastore
This MUST be returned if information identifying the reason for an
RPC error has not been placed elsewhere within the transport
portion of a failed "establish-subscription" RPC response. This
MUST be sent if hints are included.

2. yang-data modify-subscription-error-datastore
This MUST be returned if information identifying the reason for an
RPC error has not been placed elsewhere within the transport
portion of a failed "modify-subscription" RPC response. This
MUST be sent if hints are included.

3. yang-data sn:delete-subscription-error
This MUST be returned if information identifying the reason for an
RPC error has not been placed elsewhere within the transport
portion of a failed "delete-subscription" or "kill-subscription"
RPC response.

4. yang-data resynch-subscription-error
This MUST be returned if information identifying the reason for an
RPC error has not been placed elsewhere within the transport
portion of a failed "resynch-subscription" RPC response.

A.2. Notifications of Failure

A subscription may be unexpectedly terminated or suspended
independent of any RPC or configuration operation. In such cases,
indications of such a failure MUST be provided. To accomplish this,
the following types of error identities may be returned within the
corresponding subscription state change notification:

+-----------------------+-----------------------+
| subscription-terminated | subscription-suspended |
+-----------------------+-----------------------+
| datastore-not-subscribable | sn:insufficient-resources |
| sn:filter-unavailable   | period-unsupported     |
| sn:no-such-subscription | result-too-big         |
| sn:suspension-timeout   | synchronization-size   |
| unchanging-selection    |                        |

Appendix B. Changes between revisions

(To be removed by RFC editor prior to publication)

v16 - v17

- Minor updates to YANG module, incorporating comments from Tom
  Petch.
o Updated references.

v15 - v16

o Updated security considerations.

o Updated references.

o Addressed comments from last call review, specifically comments received from Martin Bjorklund.

v14 - v15

o Minor text fixes. Includes a fix to on-change update calculation to cover churn when an object changes to and from a value during a dampening period.

v13 - v14

o Minor text fixes.

v12 - v13

o Hint negotiation models now show error examples.

o yang-data structures for rpc errors.

v11 - v12

o Included Martin’s review clarifications.

o QoS moved to subscribed-notifications

o time-of-update removed as it is redundant with RFC5277’s eventTime, and other times from notification-messages.

o Error model moved to match existing implementations

o On-change notifiable removed, how to do this is implementation specific.

o NMAD model supported. Non NMAD version at https://github.com/netconf-wg/yang-push/

v10 - v11

o Promise model reference added.
- Error added for no-such-datastore
- Inherited changes from subscribed notifications (such as optional feature definitions).
-scrubbed the examples for proper encodings

v09 - v10

- Returned to the explicit filter subtyping of v00-v05
- identityref to ds:datatype made explicit
- Returned ability to modify a selection filter via RPC.

v08 - v09

- Minor tweaks cleaning up text, removing appendicies, and making reference to revised-datastores.
- Subscription-id optional in push updates, except when encoded in RFC5277, Section 4 one-way notification.
- Finished adding the text describing the resynch subscription RPC.
- Removed relationships to other drafts and future technology appendicies as this work is being explored elsewhere.
- Deferred the multi-line card issue to new drafts
- Simplified the NACM interactions.

v07 - v08

- Updated YANG models with minor tweaks to accommodate changes of ietf-subscribed-notifications.

v06 - v07

- Clarifying text tweaks.
- Clarification that filters act as selectors for subscribed datastore nodes; support for value filters not included but possible as a future extension
- Filters don’t have to be matched to existing YANG objects

v05 - v06
Internet-Draft                YANG-Push                  July 2018

- Security considerations updated.
- Base YANG model in [subscribe] updated as part of move to identities, YANG augmentations in this doc matched up
- Terms refined and text updates throughout
- Appendix talking about relationship to other drafts added.
- Datastore replaces stream
- Definitions of filters improved

v04 to v05

- Referenced based subscription document changed to Subscribed Notifications from 5277bis.
- Getting operational data from filters
- Extension notifiable-on-change added
- New appendix on potential futures. Moved text into there from several drafts.
- Subscription configuration section now just includes changed parameters from Subscribed Notifications
- Subscription monitoring moved into Subscribed Notifications
- New error and hint mechanisms included in text and in the yang model.
- Updated examples based on the error definitions
- Groupings updated for consistency
- Text updates throughout

v03 to v04

- Updates-not-sent flag added
- Not notifiable extension added
- Dampening period is for whole subscription, not single objects
- Moved start/stop into rfc5277bis
o Client and Server changed to subscriber, publisher, and receiver
o Anchor time for periodic
o Message format for synchronization (i.e. synch-on-start)
o Material moved into 5277bis
o QoS parameters supported, by not allowed to be modified by RPC
o Text updates throughout

Authors’ Addresses

Alexander Clemm
Huawei

Email: ludwig@clemm.org

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Alberto Gonzalez Prieto
VMware

Email: agonzalezpri@vmware.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

Einar Nilsen-Nygåard
Cisco Systems

Email: einarnn@cisco.com

Andy Bierman
YumaWorks

Email: andy@yumaworks.com
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:

- "TBD1" --> the assigned value for id-ct-zerotouchInformationXML
- "TBD2" --> the assigned value for id-ct-zerotouchInformationJSON

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

- "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:

- "2018-06-05" --> the publication date of this draft
The following one Appendix section is to be removed prior to publication:

- Appendix A. 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 December 7, 2018.

Copyright Notice

Copyright (c) 2018 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 .............................................. 4  
   1.1. Use Cases ........................................... 5  
   1.2. Terminology ......................................... 5  
   1.3. Requirements Language .............................. 7  
   1.4. Tree Diagrams ...................................... 7  
2. Types of Bootstrapping Information ..................... 8  
   2.1. Redirect Information ............................. 8  
   2.2. Onboarding Information ............................. 8  
3. Artifacts ............................................... 9
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

- 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.

- 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 (zero touch 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 zero touch 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.

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 "zero touch 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 zero touch 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 "zero touch 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 zero touch 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 zero touch information that has not been signed.

Zero Touch Information: The term "zero touch information" is used herein to refer either redirect information or onboarding information. Zero touch information is one of the three bootstrapping artifacts described in Section 3.

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 Bootstrapping Information

This document defines two types of information that devices can access during the bootstrapping process. These information types are described in this section. Examples are provided in Section 6.2

2.1. Redirect Information

Redirect information directs 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)
   +-- 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 information specifies how it provides the artifacts defined in this section (see Section 4).

3.1. Zero Touch Information

The zero touch 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 zero touch 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 zero touch 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 zero touch 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, or the OID id_data (1.2.840.113549.1.7.1), in which case the encoding (JSON, XML, etc.) SHOULD be communicated externally. In either case, the associated content is an octet string containing 'zerotouch-information’ data in the expected encoding.

When the zero touch 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), and its inner eContentType MUST be one of the OIDs described in Section 10.3, or the OID id_data (1.2.840.113549.1.7.1), in which case the encoding (JSON, XML, etc.) SHOULD be communicated externally. In either case, the associated content or eContent is an octet string containing ‘zerotouch-information’ data in the expected encoding.

When the zero touch 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), and 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, or the OID id_data (1.2.840.113549.1.7.1), in which case the encoding (JSON, XML, etc.) SHOULD be communicated externally. In either case, the associated content or eContent is an octet string containing ‘zerotouch-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 zero touch 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 zero touch 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), in which case 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), in which case 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 zerotouch-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 Information:** This grouping is useful for cases when transport level security can be used to convey trust (e.g., HTTPS), or when the information can be processed in a provisional manner (i.e. unsigned redirect information).

- **Signed Information, without revocations:** This grouping is useful when signed information is needed, because it is obtained from an untrusted source, and it cannot be processed provisionally, and yet either revocations are not needed or they can be obtained dynamically.

- **Signed Information, with revocations:** This grouping is useful when signed information is needed, because it is obtained from an untrusted source, and it cannot be processed provisionally, and revocations are needed and cannot be obtained dynamically.

The artifacts associated with these groupings are described below:

<table>
<thead>
<tr>
<th>Grouping</th>
<th>Zero Touch Information</th>
<th>Ownership Voucher</th>
<th>Owner Certificate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsigned Information</td>
<td>Yes, no sig</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Signed Information, with revocations</td>
<td>Yes, with sig</td>
<td>Yes, without revocations</td>
<td>Yes, without revocations</td>
</tr>
<tr>
<td>Signed Information, without revocations</td>
<td>Yes, with sig</td>
<td>Yes, without revocations</td>
<td>Yes, without revocations</td>
</tr>
</tbody>
</table>
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 zero touch 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:

Zero Touch Information: Mapped to a file containing the binary artifact described in Section 3.1 (e.g., zerotouch-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 that 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 zero touch 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.

To use a DNS server as a source of bootstrapping data, a device MAY perform a multicast DNS [RFC6762] query searching for the service "_zerotouch._tcp.local.". Alternatively the device MAY perform DNS-SD [RFC6763] via normal DNS operation, using the domain returned to it from the DHCP server; for example, searching for the service "_zerotouch._tcp.example.com".

Unsigned DNS records (e.g., not using DNSSEC as described in [RFC6698]) are an untrusted source of bootstrapping data. This means that the information stored in the DNS records either MUST be signed, or MUST be information that can be processed provisionally (e.g., unsigned redirect information).

From an artifact perspective, since a DNS server presents resource records (Section 3.2.1 of [RFC1035]), the bootstrapping artifacts need to be presented as resource records. The three artifacts defined in Section 3 are mapped to resource records below.

Artifact to Resource Record Mapping:

**Zero Touch Information:** Mapped to a TXT record called "zt-info" containing the base64-encoding of the binary artifact described in Section 3.1.
Owner Certificate: Mapped to a TXT record called "zt-cert" containing the base64-encoding of the binary artifact described in Section 3.2.

Ownership Voucher: Mapped to a TXT record called "zt-voucher" containing the base64-encoding of the binary artifact described in Section 3.3.

TXT records have an upper size limit of 65535 bytes (Section 3.2.1 in RFC1035), since "RDLENGTH" is a 16-bit field. Please see Section 3.1.3 in RFC4408 for how a TXT record can achieve this size. Due to this size limitation, some zero touch information artifacts may not fit. In particular, onboarding information could hit this upper bound, depending on the size of the included configuration and scripts.

4.3. DHCP Server

A DHCP server MAY be used as a source of zero touch 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:
Zero Touch 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 zero touch 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 information returned from it MAY be signed. However, when the server is untrusted, in order for its information to be of any use to the device, the bootstrap 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:

Zero Touch Information: Mapped to the ‘zerotouch-information’ leaf in the output of the ‘get-bootstrapping-data’ RPC.
Zero touch 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

```xml
<device>
  <read/write storage>
    1. flag to enable zerotouch bootstrapping set to "true"
  </read/write storage>
  <read-only storage>
    2. TLS client cert & related intermediate certificates
    3. list of trusted well-known bootstrap servers
    4. list of trust anchor certs for bootstrap servers
    5. list of trust anchor certs for ownership vouchers
  </read-only storage>
  <secure storage>
    6. private key for TLS client certificate
    7. private key for decrypting zerotouch artifacts
  </secure storage>
</device>
```

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 zero touch bootstrapping. This variable MUST be enabled by default in order for zero touch bootstrapping to run when the device first powers on. Because it is a goal that the configuration installed by the bootstrapping process disables zero touch 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
Devices that support decrypting zerotouch 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 (e.g., a TPM). 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.

\[
\begin{align*}
\text{Power On} & \quad | \\
\quad & | \quad | \quad | \quad | \quad | \\
& \quad | \quad | \quad | \quad | \quad | \\
\text{No} & \quad | \quad | \quad | \quad | \\
1. \text{Zerotouch bootstrapping configured} & \rightarrow \text{Boot normally} & \quad | \quad | \quad | \quad | \quad | \\
2. \text{For each supported source of bootstrapping data, try to load bootstrapping data from the source} & \quad | \quad | \quad | \quad | \quad | \\
3. \text{Able to bootstrap from any source?} & \rightarrow \text{Run with new config} & \quad | \quad | \quad | \quad | \quad | \\
4. \text{Loop and/or wait for manual provisioning.} & \quad | \quad | \quad | \quad | \quad | \\
\end{align*}
\]

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 zerotouch bootstrapping is configured, as is expected to be the case for the device’s preconfigured initial state. If zerotouch 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 MAY loop back through the list of bootstrapping sources again and/or wait for manual provisioning.

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 information. An expression that captures all possible successful sequences of bootstrapping information 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:

<table>
<thead>
<tr>
<th>Kind of Bootstrapping Data</th>
<th>Untrusted Source Can Provide?</th>
<th>Trusted Source Can Provide?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signed Redirect Info</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Signed Redirect Info</td>
<td>Yes</td>
<td>Yes*</td>
</tr>
<tr>
<td>Signed Onboarding Info</td>
<td>Yes</td>
<td>Yes*</td>
</tr>
<tr>
<td>Unsigned Redirect Info</td>
<td>Yes+</td>
<td>Yes</td>
</tr>
<tr>
<td>Unsigned Onboarding Info</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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 trusted bootstrap server (i.e. trust-state is TRUE), the device MAY, per Section 2.5 in [RFC8040], identify and authenticate itself to the bootstrap server using a TLS-level client certificate and/or an HTTP authentication scheme. If both mechanisms are used, they MUST both identify the same device using its serial number.

When establishing a connection to an untrusted bootstrap server (i.e. trust-state is FALSE), it is still necessary for the device to identify itself, in order to receive device-specific signed data, due to the ownership voucher encoding the device’s serial number. The device MUST identify and authenticate itself to the bootstrap server using a TLS-level client certificate and/or an HTTP authentication scheme. However, because the bootstrap server is untrusted, the device MUST NOT use an authentication scheme that conveys a shared secret, such as a password.

When sending a client certificate, the device MUST also send all 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 zero touch 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 bootstrap server.

If the zero touch 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 state machine described in Section 5.2. Otherwise, the device MUST attempt to process the onboarding information as described in Section 5.6. In either case, success or failure, the device MUST exit the recursive algorithm, returning to the state machine 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.
If the zero touch information artifact contains redirect information, the device MUST process the redirect information as described in Section 5.5. 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 that is all redirect information is able to redirect a device to.

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 the revocation status is not attainable or if it is determined that a certificate has been revoked, the device MUST NOT validate the owner certificate.
Finally the device MUST verify the zero touch 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. If the device encounters an error at any step, it MUST NOT proceed to the next step.

When the onboarding information is obtained from a trusted bootstrap server, the device SHOULD send progress reports throughout the bootstrapping process using the bootstrap server’s ‘report-progress’ RPC. When the onboarding information was obtained from an untrusted bootstrap server, the device SHOULD NOT send any progress reports to the bootstrap server, even after validating any signed data it may have receive from the bootstrap server.

If boot image criteria is specified, the device MUST first determine if the boot image it is running satisfies the specified boot image criteria. If the device is not running the specified boot image, then it MUST install the specified boot image or fail processing the onboarding information. In order to install the specified boot image, the device MUST download, verify, and install the specified boot image, and then reboot. To verify the downloaded boot image, the device MUST check that the boot image file matches the verification fingerprint supplied by the onboarding information. Upon rebooting, the bootstrapping process runs again, which will eventually come to this very point, but this time the device will be running the specified boot image, and thus will move to processing the next step.

Next, for devices that support executing scripts, if a pre-configuration script has been specified, the device MUST execute the script and check its exit status code to determine if it had any warnings or errors. In the case of errors, the device MUST reset itself in such a way that wipes out any bad state the script may have left behind.

Next, if an initial configuration has been supplied, the device MUST commit the provided initial configuration, using the approach specified by the ‘configuration-handling’ leaf. If there is an error, and the device previously executed a pre-configuration script, the device does not need to reset itself in order to wipe out any state the script may have left behind; this implies that the pre-configuration script must be idempotent.

Again, for devices that support executing scripts, if a post-configuration script has been specified, the device MUST execute the script and check its exit status code to determine if it had any warnings or errors. In the case of errors, the device MUST reset itself in such a way that wipes out any bad state the script may have left behind.
At this point, the device has completely processed the bootstrapping data. If the device obtained the onboarding information from a trusted bootstrap server, the device MUST post the ‘bootstrap-complete’ progress report now, using the bootstrap server’s ‘report-progress’ RPC.

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 a call home connection at this time.

6. The Zero Touch Information Data Model

This section defines a YANG 1.1 [RFC7950] module that is used to define the data model for the zero touch information artifact described in Section 3.1. This data model uses the ‘yang-data’ extension statement defined in [I-D.ietf-netmod-yang-data-ext]. 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 zero touch information artifact.

```
module:ietf-zerotouch-information

yang-data zerotouch-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-zerotouch-information:redirect-information": {
    "bootstrap-server": [
      {
        "address": "phs1.example.com",
        "port": 8443,
        "trust-anchor": "base64encodedvalue="
      },
      {
        "address": "phs2.example.com",
        "port": 8443,
        "trust-anchor": "base64encodedvalue="
      },
      {
        "address": "phs3.example.com",
        "port": 8443,
        "trust-anchor": "base64encodedvalue="
      }
    ]
  }
}
```

The following example illustrates how onboarding information (Section 2.2) can be encoded using JSON.
6.3. YANG Module

The zero touch 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 [I-D.ietf-netmod-yang-data-ext], and an encoding defined in [ITU.X690.2015].

<CODE BEGINS> file "ietf-zerotouch-information@2018-06-05.yang"

module ietf-zerotouch-information {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-zerotouch-information";
  prefix zti;

  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-yang-data-ext {

prefix yd;
reference "I-D.ietf-netmod-yang-data-ext: YANG Data Extensions";
}

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 Zero Touch Information artifact defined in RFC XXXX: Zero Touch Provisioning for Networking Devices.

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in the module text are to be interpreted as described in RFC 2119.

Copyright (c) 2018 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 2018-06-05 {
  description
    "Initial version";
  reference
    "RFC XXXX: Zero Touch Provisioning for Networking Devices";
}

// 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
yd:yang-data "zerotouch-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: Zero Touch Provisioning for Networking Devices";
            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
}

}

container onboarding-information {
  description
  "Onboarding information is described in Section 2.2 in RFC XXXXX. Its purpose is to provide the device everything
it needs to bootstrap itself.

reference
"RFC XXXX: Zero Touch Provisioning for Networking Devices"

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;
    must '../image-verification' {
      description
      "Image verification information must be provided if
      the device is going to download an image."
    }

    ordered-by user;
    description
    "An ordered list of URIs to where the necessary
    boot-image file may be obtained. Deployments must
    know through out-of-band means which URI schemes
    (http, ftp, etc.) the bootstrapping device supports.
    If a secure scheme (e.g., https) is provided, a
device MAY establish an untrusted connection to the
    remote server 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 a script is erroneously provided to a device that does not
  support the execution of scripts, and the device obtained the
  onboarding information from a trusted bootstrap server, then
  the device SHOULD send either a 'pre-script-warning' or
  'post-script-warning' progress report, based on which kind
  of script was presented, but otherwise continue processing
  the bootstrapping data as if the script had not been present.

  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 device obtained the bootstrap information from a trusted
  bootstrap server, it SHOULD either send a 'pre-script-warning'
  or 'post-script-warning' progress report, based on which kind
  of script was executed.

  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. If the device obtained
  the bootstrap information from a trusted bootstrap server, it
  SHOULD send a 'pre-script-error' or 'post-script-error'
progress report, based on which kind of script was executed, followed by a reset that will wipe out any bad state left by the script, and restart the entire bootstrapping process."

7. The Zero Touch 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-zerotouch-bootstrap-server
rpcs:
  +--x get-bootstrapping-data
    +--w input
      |   +--w untrusted-connection?   empty
      |   +--w hw-model?               string
      |   +--w os-name?                string
      |   +--w os-version?             string
      |   +--w nonce?                  binary
    +--ro output
      |   +--ro zerotouch-information    cms
      |   +--ro owner-certificate?       cms
      |   +--ro ownership-voucher?       cms
    +--x report-progress
      +--w input
        |   +--w progress-type           enumeration
        |   +--w message?                string
        |   +--w ssh-host-keys
        |      +--w ssh-host-key*   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 a untrusted bootstrap server. In this example, the device sends the ‘untrusted-connection’ input parameter and receives signed data in the response.

REQUEST
-------
[‘\’ line wrapping added for formatting only]

POST /restconf/operations/ietf-zerotouch-bootstrap-server:get-boot\strapping-data HTTP/1.1
HOST: example.com
Content-Type: application/yang.data+xml

<input
xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server">
<untrusted-connection/>
</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-zerotouch-bootstrap-server">
<zerotouch-information>base64encodedvalue==</zerotouch-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.
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
-------
[’\’ line wrapping added for formatting only]

POST /restconf/operations/ietf-zerotouch-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-zerotouch-bootstrap-server">
  <progress-type>bootstrap-complete</progress-type>
  <message>example message</message>
  <ssh-host-keys>
    <ssh-host-key>base64encodedvalue==</ssh-host-key>
    <ssh-host-key>base64encodedvalue2==</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], and uses an encoding defined in [ITU.X690.2015].

<CODE BEGINS> file "ietf-zerotouch-bootstrap-server@2018-06-05.yang"
module ietf-zerotouch-bootstrap-server {
  yang-version 1.1;
  namespace
  prefix ztbs;
  organization
    "IETF NETCONF (Network Configuration) Working Group";

description


Copyright (c) 2018 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 2018-06-05 {
    description
        "Initial version";
    reference
        "RFC XXXX: Zero Touch Provisioning for Networking Devices";
}

// 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),

Watsen, et al. Expires December 7, 2018
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 untrusted-connection {
      type empty;
      description
          "This optional input parameter enables a device to communicate to the bootstrap server that it is unable to authenticate the bootstrap server’s TLS certificate. In such circumstances, the device likely does not send any of the other input parameters, except for the 'nonce' parameter. Upon receiving this input parameter, the bootstrap server should only return unsigned redirect information or signed data of any type.";
    }
    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
    }
    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 "8..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 zerotouch-information {

type cms;

mandatory true;

description
"A zero touch information artifact, as described in Section 3.1 of RFC XXXX."

reference
"RFC XXXX:
Zero Touch Provisioning for Networking Devices"

}

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 zero touch information artifact"
is signed.
reference
"RFC XXXX:
Zero Touch Provisioning for Networking Devices";
}
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 zero touch information artifact
  is signed.";
  reference
  "RFC XXXX:
  Zero Touch Provisioning for Networking Devices";
}
}
}

rpc report-progress {
  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-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 "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 "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 "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 "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 "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.

```yaml
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 trust anchor certificates an NMS may use to authenticate subsequent SSH-based connections to this device (e.g., netconf-ssh, netconf-ch-ssh).";
}
leaf-list ssh-host-key {
  type binary;
  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 device’s end-entity certificate 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 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:
RFC 5652:
Cryptographic Message Syntax (CMS)";
}
}
}

<CODE ENDS>
8. DHCP Zero Touch 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 Zero Touch Option

The DHCPv4 Zero Touch Option is used to provision the client with one
or more URIs for bootstrap servers that can be contacted to attempt
further configuration.

DHCPv4 Zero Touch Redirect Option

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) .
.
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

o option-code: OPTION_V4_ZEROTOUCH_REDIRECT (143)
o option-length: The option length in octets
o 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 [common-field-encoding].

DHCPv4 Client Behavior

Clients MAY request the OPTION_V4_ZEROTOUCH_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_ZEROTOUCH_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_ZEROTOUCH_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 ZERO TOUCH_REDIRECT option instances.

DHCPv4 Server Behavior

The DHCPv4 server MAY include a single instance of Option OPTION_V4 ZERO TOUCH_REDIRECT in DHCP messages it sends. Servers MUST NOT send more than one instance of the OPTION_V4 ZERO TOUCH_REDIRECT option.

As the list of URIs may exceed the maximum allowed length of a single DHCPv4 option (255 octets), the server MUST implement [RFC3396], allowing the URI list to be split across a number of OPTION_V4 ZERO TOUCH_REDIRECT option instances.

8.2. DHCPv6 Zero Touch Option

The DHCPv6 Zero Touch Option is used to provision the client with one or more URIs for bootstrap servers that can be contacted to attempt further configuration.

DHCPv6 Zero Touch Redirect Option

<table>
<thead>
<tr>
<th>option-code (136)</th>
<th>option-length</th>
<th>bootstrap-server-list (variable length)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+++++++++++++++++++</td>
<td>+---------------</td>
<td>+----------------------------------------</td>
</tr>
<tr>
<td>option-code: OPTION_V6 ZERO TOUCH_REDIRECT (136)</td>
<td>option-length: The option length in octets</td>
<td>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 [common-field-encoding].</td>
</tr>
</tbody>
</table>

DHCPv6 Client Behavior

Clients MAY request the OPTION_V6 ZERO TOUCH_REDIRECT option, as defined in [RFC3315], Sections 17.1.1, 18.1.1, 18.1.3, 18.1.4, 18.1.5, and 22.7. 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_ZEROTOUCH_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_ZEROTOUCH_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

Sections 17.2.2 and 18.2 of [RFC3315] govern 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_ZEROTOUCH_REDIRECT is a singleton. Servers MUST NOT send more than one instance of the OPTION_V6_ZEROTOUCH_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 an option that can contain multiple URIs (see [RFC7227], Section 5.7).

    bootstrap-server-list:
    +---------------------------+-----------------------------+
    | uri-length                | URI                         |
    +---------------------------+-----------------------------+

    o uri-length: variable, in octets.
    o URI: URI of zerotouch bootstrap server, using the HTTPS URI scheme defined in Section 2.7.2 of RFC7230. URI MUST be in form "https://<ip-address-or-hostname>[:<port>]".

9. Security Considerations

9.1. 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.

9.2. Secure Storage for Long-lived Private Keys

Manufacturer-generated device identifiers may have very long lifetimes. For instance, [Std-802.1AR-2009] 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 (e.g., a TPM).

9.3. Use of IDevID Certificates

IDevID certificates, as defined in [Std-802.1AR-2009], 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 zerotouch artifacts.

9.4. 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.

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 enables devices to establish provisional connections to bootstrap servers, in order for the bootstrap server to provide either unsigned redirect information or signed data of any type to the device. However, since the server is untrusted, it may be under the control of an adversary, and therefore devices should be cautious about the data they send in such cases.

This document requires devices identify and authenticate themselves to untrusted bootstrap servers. Depending on the authentication mechanisms used, this means that, at a minimum, the device’s serial number may be disclosed to an adversary. Serial numbers are ubiquitous and prominently contained in invoices and on labels affixed to devices and their packaging. That said, serial numbers many times encode revealing information, such as the device’s model number, manufacture date, and/or manufacturing sequence number. Knowledge of this information may provide an adversary with details needed to launch an attack.

In addition to the information relayed during the authentication, other potentially identifying values that may be disclosed to an untrusted server, including ‘os-name’, ‘os-version’, ‘hw-model’, and progress reports. In order to address this issue, it is RECOMMENDED that bootstrap server implementations promote the untrusted connection to a trusted connection, as described in Appendix B.

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. The "ietf-zerotouch-information" YANG Module

The ietf-zerotouch-information 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, 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-zerotouch-information 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 contents of the CMS structure are redirect-information, an observer can learn about the bootstrap servers the device is being directed, 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 contents of the CMS structure are onboarding-information, an observer could learn considerable information about how the device is to be provisioned. This information includes the specific operating system version, the initial configuration, and the specific scripts that the device is to run. All of this information should be considered highly sensitive and precautions should be taken to protect it from falling into the wrong hands.

9.9. The "ietf-zerotouch-bootstrap-server" YANG Module

The ietf-zerotouch-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 [RFC5246].

The NETCONF Access Control Model (NACM) [RFC6536] 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 IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:

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

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

10.2. The YANG Module Names Registry

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

name: ietf-zerotouch-information
prefix: zti
reference: RFC XXXX

name: ietf-zerotouch-bootstrap-server
namespace: urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server (note: '\' used for formatting reasons only)
prefix: ztbs
reference: RFC XXXX

10.3. The SMI Security for S/MIME CMS Content Type Registry

IANA is kindly requested to two entries in the "SMI Security for S/MIME CMS Content Type" registry (1.2.840.113549.1.9.16.1), with values as follows:
<table>
<thead>
<tr>
<th>Decimal</th>
<th>Description</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD1</td>
<td>id-ct-zerotouchInformationXML</td>
<td>[RFCXXXX]</td>
</tr>
<tr>
<td>TBD2</td>
<td>id-ct-zerotouchInformationJSON</td>
<td>[RFCXXXX]</td>
</tr>
</tbody>
</table>

10.4. The BOOTP Manufacturer Extensions and DHCP Options Registry

IANA is kindly requested to make permanent the following early code point allocation in the "BOOTP Manufacturer Extensions and DHCP Options" registry maintained at http://www.iana.org/assignments/bootp-dhcp-parameters:

Tag: 143  
Name: OPTION_V4_ZEROTOUCH_REDIRECT  
Data Length: N  
Meaning: This option provides a list of URIs for zerotouch bootstrap servers  
Reference: [RFCXXXX]

And the following early code point allocation in the "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)" registry maintained at http://www.iana.org/assignments/dhcpv6-parameters:

Value: 136  
Description: OPTION_V6_ZEROTOUCH_REDIRECT  
Reference: [RFCXXXX]

11. References

11.1. Normative References

[I-D.ietf-netmod-yang-data-ext]  

[ITU.X690.2015]  

[RFC1035]  


11.2. Informative References

[I-D.ietf-netconf-crypto-types]
Watsen, K., "Common YANG Data Types for Cryptography", draft-ietf-netconf-crypto-types-00 (work in progress), June 2018.
[I-D.ietf-netconf-trust-anchors]
Watsen, K., "YANG Data Model for Global Trust Anchors",
draft-ietf-netconf-trust-anchors-00 (work in progress),
June 2018.

[RFC3688]  Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688,
DOI 10.17487/RFC3688, January 2004,

(TLS) Protocol Version 1.2", RFC 5246,
DOI 10.17487/RFC5246, August 2008,

and A. Bierman, Ed., "Network Configuration Protocol
(NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011,

Protocol (NETCONF) Access Control Model", RFC 6536,
DOI 10.17487/RFC6536, March 2012,

[RFC6698]  Hoffman, P. and J. Schlyter, "The DNS-Based Authentication
of Named Entities (DANE) Transport Layer Security (TLS)

[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,

[RFC8071]  Watsen, K., "NETCONF Call Home and RESTCONF Call Home",
RFC 8071, DOI 10.17487/RFC8071, February 2017,

BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018,
Appendix A. The Zero Touch Device Data Model

This section defines a non-normative data model that enables the configuration of zero-touch 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 zero-touch device data model.

```plaintext
module: example-zerotouch-device
   +--rw zerotouch
       +--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-pinned-certificates?
               ta:pinned-certificates-ref (bootstrap-servers)?
               +--ro voucher-pinned-certificates?
                   ta:pinned-certificates-ref (signed-data)?
```

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 zero-touch bootstrapping is longer needed.

A.2. Example Usage

Following is an instance example for this data model.
<zerotouch
xmlns="https://example.com/zerotouch-device">
<enabled>true</enabled>
<idevid-certificate>base64encodedvalue==</idevid-certificate>
<bootstrap-servers>
  <bootstrap-server>
    <address>phs1.example.com</address>
    <port>8443</port>
  </bootstrap-server>
  <bootstrap-server>
    <address>phs2.example.com</address>
    <port>8443</port>
  </bootstrap-server>
  <bootstrap-server>
    <address>phs3.example.com</address>
    <port>8443</port>
  </bootstrap-server>
</bootstrap-servers>
<bootstrap-server-pinned-certificates>manufacturers-root-ca-certs</bootstrap-server-pinned-certificates>
<voucher-pinned-certificates>manufacturers-root-ca-certs</voucher-pinned-certificates>
</zerotouch>

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-zerotouch-device {
  yang-version 1.1;
  namespace "https://example.com/zerotouch-device";
  prefix ztd;

  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 zerotouch 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 2018-06-05 {
  description
  "Initial version";
  reference
  "RFC XXXX: Zero Touch Provisioning for Networking Devices";
}

// 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 zerotouch {
  description
  "Top-level container for zerotouch data model.";
  leaf enabled {
    type boolean;
    default false;
    description
    "The 'enabled' leaf controls if zerotouch 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."
}

leaf bootstrap-server-pinned-certificates {
  if-feature bootstrap-servers;
  type ta:pinned-certificates-ref;
  config false;
  description "A reference to a list of pinned certificate authority (CA) certificates that the device uses to validate bootstrap servers with."
}

leaf voucher-pinned-certificates {
  if-feature signed-data;
  type ta:pinned-certificates-ref;
  config false;
  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 ‘untrusted-connection’ input parameter, and optionally also the ‘nonce’ input parameter, in the ‘get-bootstrapping-data’ RPC. The ‘untrusted-connection’ 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 ‘untrusted-connection’ 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 ‘untrusted-connection’ parameter was not passed, having access to all of the device’s input parameters, the bootstrap server returns unsigned onboarding information to the device.

Appendix C. Workflow Overview

The zero touch 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 zero touch 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’re 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 zero touch 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 information 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 information 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 information 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 information, a DNS server needs to be configured. If multicast DNS-SD is desired, then the 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 information, the information 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.

```
1. if zerotouch bootstrap service is not enabled, then exit.
2. for each source supported, check for bootstrapping data.
3. if onboarding information found, initialize self and, only if source is a bootstrap server, send progress updates.
4. else if redirect-information found, for each bootstrap server specified, check for data.
   if more redirect-information is found, recurse (not depicted), else if onboarding-information found, initialize self and post progress reports
5. retry sources and/or wait for manual provisioning.
```

The interactions in the above diagram are described below.

1. Upon power being applied, the device checks to see if zerotouch bootstrapping is configured, such as must be the case when running its "factory default" configuration. If zerotouch
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-rsa 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 zerotouch 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 it 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 zerotouch bootstrapping service, as the need for bootstrapping would no longer be present.

Appendix D. Change Log

D.1. ID to 00

- Major structural update; the essence is the same. Most every section was rewritten to some degree.
- Added a Use Cases section
- Added diagrams for "Actors and Roles" and "NMS Precondition" sections, and greatly improved the "Device Boot Sequence" diagram
- Removed support for physical presence or any ability for configlets to not be signed.
- Defined the Zero Touch Information DHCP option
- Added an ability for devices to also download images from configuration servers
- Added an ability for configlets to be encrypted
- Now configuration servers only have to support HTTP/S – no other schemes possible

D.2. 00 to 01

- Added boot-image and validate-owner annotations to the "Actors and Roles" diagram.
- Fixed 2nd paragraph in section 7.1 to reflect current use of anyxml.
- Added encrypted and signed-encrypted examples
- Replaced YANG module with XSD schema
- Added IANA request for the Zero Touch Information DHCP Option
- Added IANA request for media types for boot-image and configuration
D.3. 01 to 02

- Replaced the need for a configuration signer with the ability for each NMS to be able to sign its own configurations, using manufacturer signed ownership vouchers and owner certificates.
- Renamed configuration server to bootstrap server, a more representative name given the information devices download from it.
- Replaced the concept of a configlet by defining a southbound interface for the bootstrap server using YANG.
- Removed the IANA request for the boot-image and configuration media types.

D.4. 02 to 03

- Minor update, mostly just to add an Editor’s Note to show how this draft might integrate with the draft-pritikin-anima-bootstrapping-keyinfra.

D.5. 03 to 04

- Major update formally introducing unsigned data and support for Internet-based redirect servers.
- Added many terms to Terminology section.
- Added all new "Guiding Principles" section.
- Added all new "Sources for Bootstrapping Data" section.
- Rewrote the "Interactions" section and renamed it "Workflow Overview".

D.6. 04 to 05

- Semi-major update, refactoring the document into more logical parts
- Created new section for information types
- Added support for DNS servers
- Now allows provisional TLS connections
- Bootstrapping data now supports scripts
Device Details section overhauled

Security Considerations expanded

Filled in enumerations for notification types

D.7. 05 to 06

Minor update

Added many Normative and Informative references.

Added new section Other Considerations.

D.8. 06 to 07

Minor update

Added an Editorial Note section for RFC Editor.

Updated the IANA Considerations section.

D.9. 07 to 08

Minor update

Updated to reflect review from Michael Richardson.

D.10. 08 to 09

Added in missing "Signature" artifact example.

Added recommendation for manufacturers to use interoperable formats and file naming conventions for removable storage devices.

Added configuration-handling leaf to guide if config should be merged, replaced, or processed like an edit-config/yang-patch document.

Added a pre-configuration script, in addition to the post-configuration script from -05 (issue #15).

D.11. 09 to 10

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’s encoded, matching the language that was in Appendix B.
D.14. 12 to 13

- Defined a standalone artifact to encode the old information-type into a PKCS #7 structure.
- Standalone information artifact hardcodes JSON encoding (to match the voucher draft).
- Combined the information and signature PKCS #7 structures into a single PKCS #7 structure.
- Moved the certificate-revocations into the owner-certificate’s PKCS #7 structure.
- Eliminated support for voucher-revocations, to reflect the voucher-draft’s switch from revocations to renewals.

D.15. 13 to 14

- Renamed "bootstrap information" to "onboarding information".
- Rewrote DHCP sections to address the packet-size limitation issue, as discussed in Chicago.
- Added Ian as an author for his text-contributions to the DHCP sections.
- Removed the Guiding Principles section.

D.16. 14 to 15

- Renamed action ‘notification’ to ‘update-progress’ and, likewise ‘notification-type’ to ‘update-type’.
- Updated examples to use "base64encodedvalue==" for binary values.
- Greatly simplified the "Artifact Groupings" section, and moved it as a subsection to the "Artifacts" section.
- Moved the "Workflow Overview" section to the Appendix.
- Renamed "bootstrap information" to "update information".
- Removed "Other Considerations" section.
- Tons of editorial updates.
D.17. 15 to 16

- Tweaked language to refer to "initial state" rather than "factory default configuration", so as accommodate white-box scenarios.
- Added a paragraph to Intro regarding how the solution primarily regards physical machines, but could be extended to VMs by a future document.
- Added a pointer to the Workflow Overview section (recently moved to the Appendix) to the Intro.
- Added a note that, in order to simplify the verification process, the "Zerotouch Information" PKCS #7 structure MUST also contain the signing X.509 certificate.
- Noted that the owner certificate’s must either have no Key Usage or the Key Usage must set the "digitalSignature" bit.
- Noted that the owner certificate’s subject and subjectAltName values are not constrained.
- Moved/consolidated some text from the Artifacts section down to the Device Details section.
- Tightened up some ambiguous language, for instance, by referring to specific leaf names in the Voucher artifact.
- Reverted a previously overzealous s/unique-id/serial-number/ change.
- 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

- Added an example for how to promote an untrusted connection to a trusted connection.
- Added a "query parameters" section defining some parameters enabling scenarios raised in last call.
- Added a "Disclosing Information to Untrusted Servers" section to the Security Considerations.
D.19. 17 to 18

- Added Security Considerations for each YANG module.
- Reverted back to the device always sending its DevID cert.
- Moved data tree to `ac’get-bootstrapping-data’ RPC.
- Moved the ‘update-progress’ action to a ‘report-progress’ RPC.
- Added an ‘untrusted-connection’ parameter to ‘get-bootstrapping-data’ RPC.
- Added the "ietf-zerotouch-device" module.
- Lots of small updates.

D.20. 18 to 19

- 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

- Now references I-D.ietf-netmod-yang-tree-diagrams.
- Fixed tree-diagrams in Section 2 to always reflect current YANG (now they are now dynamically generated).
- The "redirect-information" container’s "trust-anchor" is now a CMS structure that can contain a chain of certificates, rather than a single certificate.
- The "onboarding-information" container’s support for image verification reworked to be extensible.
- Added a reference to the "Device Details" section to the new example-zerotouch-device module.
- Clarified that the device must always pass its IDevID certificate, even for untrusted bootstrap servers.
- 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.
For the get-bootstrapping-data RPC’s input, removed the "remote-id" and "circuit-id" fields, and added a "hw-model" field.

Improved DHCP error handling text.

Added MUST requirement for DHCPv6 client and server implementing [RFC3396] to handle URI lists longer than 255 octets.

Changed the "configuration" value in onboarding-information to be type ‘binary’ instead of ‘anydata’.

Moved everything from PKCS#7 to CMS (this shows up as a big change).

Added the early code point allocation assignments for the DHCP Options in the IANA Considerations section, and updated the RFC Editor note accordingly.

Added RFC Editor request to replace the assigned values for the CMS content types.

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.

Moved normative module "ietf-zerotouch-device" to non-normative module "example-zerotouch-device".

Updated Title, Abstract, and Introduction per discussion on list.

D.22.  20 to 21

Now any of the three artifact can be encrypted.

Fixed some line-too-long issues.

D.23.  21 to 22

Removed specifics around how scripts indicate warnings or errors and how scripts emit output.

Moved the Zero Touch Device Data Model section to the Appendix.

Modified the YANG module in the Zero Touch Device Data Model section to reflect the latest trust-anchors and keystore drafts.
Modified types in other YANG modules to more closely emulate what is in draft-ietf-netconf-crypto-types.

Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): David Harrington, Michael Behringer, Dean Bogdanovic, Martin Bjorklund, Joe Clarke, Toerless Eckert, Stephen Farrell, Stephen Hanna, Wes Hardaker, Radek Krejci, Russ Mundy, Reinaldo Penno, Randy Presuhn, Max Pritikin, Michael Richardson, Phil Shafer, Juergen Schoenwaelder.

Special thanks goes to Steve Hanna, Russ Mundy, and Wes Hardaker for brainstorming the original I-D’s solution during the IETF 87 meeting in Berlin.

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
Subscription to Multiple Stream Originators
draft-zhou-netconf-multi-stream-originators-02

Abstract

This document describes the distributed data collection mechanism that allows multiple data streams to be managed using a single subscription. Specifically, multiple data streams are pushed directly to the collector without passing through a broker for internal consolidation.

Requirements Language

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].

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 November 17, 2018.
1. Introduction

Streaming telemetry refers to sending a continuous stream of operational data from a device to a remote receiver. This provides an ability to monitor a network from remote and to provide network analytics. Devices generate telemetry data and push that data to a collector for further analysis. By streaming the data, much better performance, finer-grained sampling, monitoring accuracy, and bandwidth utilization can be achieved than with polling-based alternatives.
YANG-Push [I-D.ietf-netconf-yang-push] defines a transport-independent subscription mechanism for datastore updates, in which a subscriber can subscribe to a stream of datastore updates from a server, or update provider. The current design involves subscription to a single push server. This conceptually centralized model encounters efficiency limitations in cases where the data sources are themselves distributed, such as line cards in a piece of network equipment. In such cases, it will be a lot more efficient to have each data source (e.g., each line card) originate its own stream of updates, rather than requiring updates to be tunneled through a central server where they are combined. What is needed is a distributed mechanism that allows to directly push multiple individual data substreams, without needing to first pass them through an additional processing stage for internal consolidation, but still allowing those substreams to be managed and controlled via a single subscription.

This document will describe such distributed data collection mechanism and how it can work by extending existing YANG-Push mechanism. The proposal is general enough to fit many scenarios.

2. Use Cases

2.1. Use Case 1: Data Collection from Devices with Main-board and Line-cards

For data collection from devices with main-board and line-cards, existing YANG-Push solutions consider only one push server typically reside in the main board. As shown in the following figure, data are collected from line cards and aggregate to the main board as one consolidated stream. So the main board can easily become the performance bottle-neck. The optimization is to apply the distributed data collection mechanism which can directly push data from line cards to a collector. On one hand, this will reduce the cost of scarce compute and memory resources on the main board for data processing and assembling. On the other hand, distributed data push can off-load the streaming traffic to multiple interface
2.2. Use Case 2: IoT Data Collection

In the IoT data collection scenario, as shown in the following figure, collector usually cannot access to IoT nodes directly, but is isolated by the border router. So the collector subscribes data from the border router, and let the border router to disassemble the subscription to corresponding IoT nodes. The border router is typically the traffic convergence point. It’s intuitive to treat the border router as a broker assembling the data collected from the IoT nodes and forwarding to the collector[I-D.ietf-core-coap-pubsub]. However, the border router is not so powerful on data assembling as a network device. It’s more efficient for the collector, which may be a server or even a cluster, to assemble the subscribed data if possible. In this case, push servers that reside in IoT nodes can stream data to the collector directly while traffic only passes through the border router.
3. Solution Overview

All the use cases described in the previous section are very similar on the data subscription and publication mode, hence can be abstracted to the following generic distributed data collection framework, as shown in the following figure.

A Collector usually includes two components,

- the Subscriber generates the subscription instructions to express what and how the collector want to receive the data;
- the Receiver is the target for the data publication.

For one subscription, there may be one to many receivers. And the subscriber does not necessarily share the same address with receivers.

In this framework, the stream originators have the Master role and the Agent role. Both the Master and the Agent include two components,

- the Subscription Server manages capabilities that it can provide to the subscriber.
- the Publisher pushes data to the receiver according to the subscription information.
The Master knows all the capabilities that the attached Agents and itself can provide, and exposes the Global Capability to the Collector. The Collector cannot see the Agents directly, so it will only send the Global Subscription information to the Master. The Master disassembles the Global Subscription to multiple Component Subscriptions, each involving data from a separate telemetry source. The Component Subscriptions are then distributed to the corresponding Agents.

When data streaming, the Publisher located in each stream originator collects and encapsulates the packets per the Component Subscription, and pushes the piece of data which can serve directly to the designated data Collector. The Collector is able to assemble many pieces of data associated with one Global Subscription, and can also deduce the missing pieces of data.

Fig. 3 The Generic Distributed Data Collection Framework

Master and Agents may interact with each other in several ways:
o Agents need to have a registration or announcement handshake with the Master, so the Master is aware of them and of life-cycle events (such as Agent appearing and disappearing).

o Contracts are needed between the Master and each Agent on the Component Capability, and the format for streaming data structure.

o The Master relays the component subscriptions to the Agents.

o The Agents indicate status of Component Subscriptions to the Master. The status of the overall subscription is maintained by the Master. The Master is also responsible for notifying the subscriber in case of any problems of Component Subscriptions.

Any technical mechanisms or protocols used for the coordination of operational information between Master and Agent is out-of-scope of the solution. We will need to instrument the results of this coordination on the Master Node.

Note: Some preliminary considerations on the solution details are now listed in the appendix for reference. The detailed solution need to be discussed and will be added if the WG accepts the problem statement.

4. Subscription Decomposition

Since Agents are invisible to the Collector, the Collector can only subscribe to the Master. This requires the Master to:

1. expose the Global Capability that can be served by multiple stream originators;

2. disassemble the Global Subscription to multiple Component Subscriptions, and distribute them to the corresponding telemetry sources;

3. notify on changes between portions of a subscription moving between different Agents over time.

To achieve the above requirements, the Master need a Global Capability description which is typically the YANG [RFC7950] data model. This global YANG model is provided as the contract between the Master and the Collector. Each Agent associating with the Master owns a local YANG model to describe the Component Capabilities which it can serve as part of the Global Capability. All the Agents need to know the namespace associated with the Master.
The Master also need a data structure, typically a Resource-Location Table, to keep track of the mapping between the resource and the corresponding location of the Subscription Server which commits to serve the data. When a Global Subscription request arrives, the Master will firstly extract the filter information from the request. Consequently, according to the Resource-Location Table, the Global Subscription can be disassembled into multiple Component Subscriptions, and the corresponding location can be associated.

The decision whether to decompose a Global Subscription into multiple Component Subscriptions rests with the Resource-Location Table. A Master can decide to not decompose a Global Subscription at all and push a single stream to the receiver, because the location information indicates the Global Subscription can be served locally by the Master. Similarly, it can decide to entirely decompose a Global Subscription into multiple Component Subscriptions that each push their own streams, but not from the Master. It can also decide to decompose the Global Subscription into several Component Subscriptions and retain some aspects of the Global Subscription itself, also pushing its own stream.

Component Subscriptions belonging to the same Global Subscription MUST NOT overlap. The combination of all Component Subscriptions MUST cover the same range of nodes as the Global Subscription. Also, the same subscription settings apply to each Component Subscription, i.e., the same receivers, the same time periods, the same encodings are applied to each Component Subscription per the settings of the Global Subscription.

Each Component Subscription in effect constitutes a full-fledged subscription, with the following constraints:

- Component subscriptions are system-controlled, i.e. managed by the Master Node, not by the subscriber.
- Component subscription settings such as time periods, dampening periods, encodings, receivers adopt the settings of their Global Subscription.
- The life-cycle of the Component Subscription is tied to the life-cycle of the Global Subscription. Specifically, terminating/removing the Global Subscription results in termination/removal of Component Subscriptions.
- The Component Subscriptions share the same Subscription ID as the Global Subscription.
5. Publication Composition

The Publisher collects data and encapsulates the packets per the component subscription. There are several potential encodings, including XML, JSON, CBOR and GPB. The format and structure of the data records are defined by the YANG schema, so that the composition at the Receiver can benefit from the structured and hierarchical data instance.

The Receiver is able to assemble many pieces of data associated with one subscription, and can also deduce the missing pieces of data. The Receiver recognizes data records associated with one subscription according the Subscription ID. Data records generated per one subscription are assigned with the same Subscription ID.

For the time series data stream, records are produced periodically from each stream originator. The message arrival time varies because of the distributed nature of the publication. The Receiver assembles data generated at the same time period based on the recording time consisted in each data record. In this case, time synchronization is required for all the steam originators.

6. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

7. Security Considerations

It’s expected to reuse the existing secure transport layer protocols, such as TLS [RFC5246] and DTLS [RFC6347], to secure the telemetry stream. The Collector cannot access the Agent directly but to negotiate the security parameters with the Master. However the data streams are actually generated by the Agents which are invisible to the Collector. So mechanisms may need to consider when adapting secure transport layer protocols here. the detailed solution is TBD.

8. Acknowledgements

9. References

9.1. Normative References
9.2. Informative References

[I-D.ietf-core-coap-pubsub]

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

Appendix A. Change Log

(To be removed by RFC editor prior to publication)
v01
o Minor revision on Subscription Decomposition
o Revised terminologies
o Removed most implementation related text
o Place holder of two sections: Subscription Management, and Notifications on Subscription State Changes

v02
Revised section 4 and 5. Moved them from appendix to the main text.

Appendix B. Subscription Management

A Global Subscription can be rejected for multiple reasons. Some are related to the Subscription Decomposition and Component Subscription. New error codes are defined to indicate why a datastore subscription attempt has failed. The subscription result with the failure reason is returned as part of the RPC response.

Appendix C. Notifications on Subscription State Changes

Each component subscription maintains its own subscription state and is responsible for sending its own OAM notifications (for example, when the component subscription is suspended or when it can resume).

TBD.

Appendix D. Configured Subscription and Call Home

TBD. Only about the message layer which is transport independent.

Authors’ Addresses

Tianran Zhou
Huawei
156 Beiqing Rd., Haidian District
Beijing
China

Email: zhoutianran@huawei.com

Guangying Zheng
Huawei
101 Yu-Hua-Tai Software Road
Nanjing, Jiangsu
China

Email: zhengguangying@huawei.com

Eric Voit
Cisco Systems
United States of America

Email: evoit@cisco.com
Alexander Clemm  
Huawei  
2330 Central Expressway  
Santa Clara, California  
United States of America  

Email: alexander.clemm@huawei.com

Andy Bierman  
YumaWorks  
United States of America  

Email: andy@yumaworks.com